USER MANUAL

LASER MICROMACHINING SYSTEM SOFTWARE
MicroMMI

Version 1.6

Software By: Dana Lee Church
DanaChurch@teosys.com

1 Overview

The MicroMMI application is a Graphical User Interface (GUI), which allows a user to program a Potomac LMT micromachining system using a superset of RS 274, or G-Code motion control programming commands, and view a live video feed set up to display aspects of the micromachining process. Proprietary machine vision commands have been added to the motion control programming commands to allow the user to make motion-based decisions from the output of real time image processing and measurement functions.

The basic MicroMMI program was designed for use in the production environment. In order ensure proper system operation, the basic application limits the user's control to those functions that are required to complete the production operation. With the addition of machine vision, MicroMMI now requires a tremendous knowledge of the specific application domain as well as the particulars of the MicroMMI application itself.

The program has been designed for use in the production environment. In order ensure proper part and system operation, MicroMMI limits the user's control to those functions that are required to complete the production operation. MicroMMI is especially useful in environments where the operator's computer skills are limited, or where process consistency is essential. Due to its self-contained and automatic nature, MicroMMI is an ideal package for ISO-9002 operations.

2 Background

The MicroMMI application consists of five main parts:

1) Video subsystem
2) Motion control subsystem
3) Laser subsystem
4) Integrated Development Environment

2.1 Video Subsystem

The video subsystem provides the visual display of the part in process. In order for the video subsystem to display a live video stream, a frame grabber card must inserted into a PCI slot of the computer on which the application runs. It is the job of the frame grabber to convert the analog video stream into a digital representation, which may then be written to the video display memory of the computer where it can be seen on the monitor. MicroMMI's video subsystem configures the frame grabber to communicate with the specific camera being used by the system. From there, the video stream may be started and stopped and graphics may be overlaid. There are many functions that can negatively affect the performance of the video including changing or removing the camera, inserting another card which conflicts with the frame grabber or removing the frame grabber altogether, or a malfunction of any of the video subsystem hardware.

2.2 Motion Control Subsystem

The motion control subsystem includes the U500 PC card that is plugged into an ISA slot in the computer, the cable that leads from the U500 card to the DR 500 amplifier, and the stages, which are connected to the amplifier. The U500 card has firmware on it, which takes a PRG program written in G-Code, and interprets it to provide the control of the motion system and stages. MicroMMI includes a visual interface that indicates the current state of the motion system including stage positions, current PRG variable values, and current IO status. The motion system must be configured using a parameter file that is specific to the particular system and application. There are other files that are necessary for the motion system to operate properly. See the Aerotech documentation for more information.

2.3 Laser Subsystem

The laser subsystem includes the laser and the operation of the laser. MicroMMI allows the user to fire the laser from the user interface and from within a PRG program. That is the extent of MicroMMI's communication with the laser subsystem. See the Potomac Operator's Manual for the laser for more information.

2.4 Integrated Development Environment

MicroMMI allows the user to load and edit any PRG program. Editing is tightly integrated with PRG program execution. From within the editor's interface, the user may run the currently loaded PRG program or single step through each line of the PRG program. Because motion programming is a real time operation, single stepping through a PRG program may significantly affect the operation of the program. It is not a good

idea to single step through a laser fire line of code as the laser will continue firing until the laser is told to stop firing by another command. The ability to single step through a PRG program allows the user to step through sections of the program to debug the control logic. The combination of the editing and the PRG program execution makes up the Integrated Development Environment (IDE) of the MicroMMI application.

MicroMMI is an Integrated Development Environment (IDE) for PRG (G-Code) programming. An IDE allows, among other things, the loading, editing, and debugging of PRG (G-Code) programs similar in nature to what the Visual Basic IDE provides for Visual Basic programmers. As such, there are many ways that the environment may be used that will affect the execution of the PRG program. The following will provide an overview of the programming process with an emphasis on how using MicroMMI may differ from the 'normal' programming process.

The steps involved in the average programmer's development process are the following:

1) Load existing program or existing program which is similar to the desired functionality
2) Edit the loaded program to contain the new desired functionality
3) Run the program to see what it does
4) Edit the program to change undesired functionality

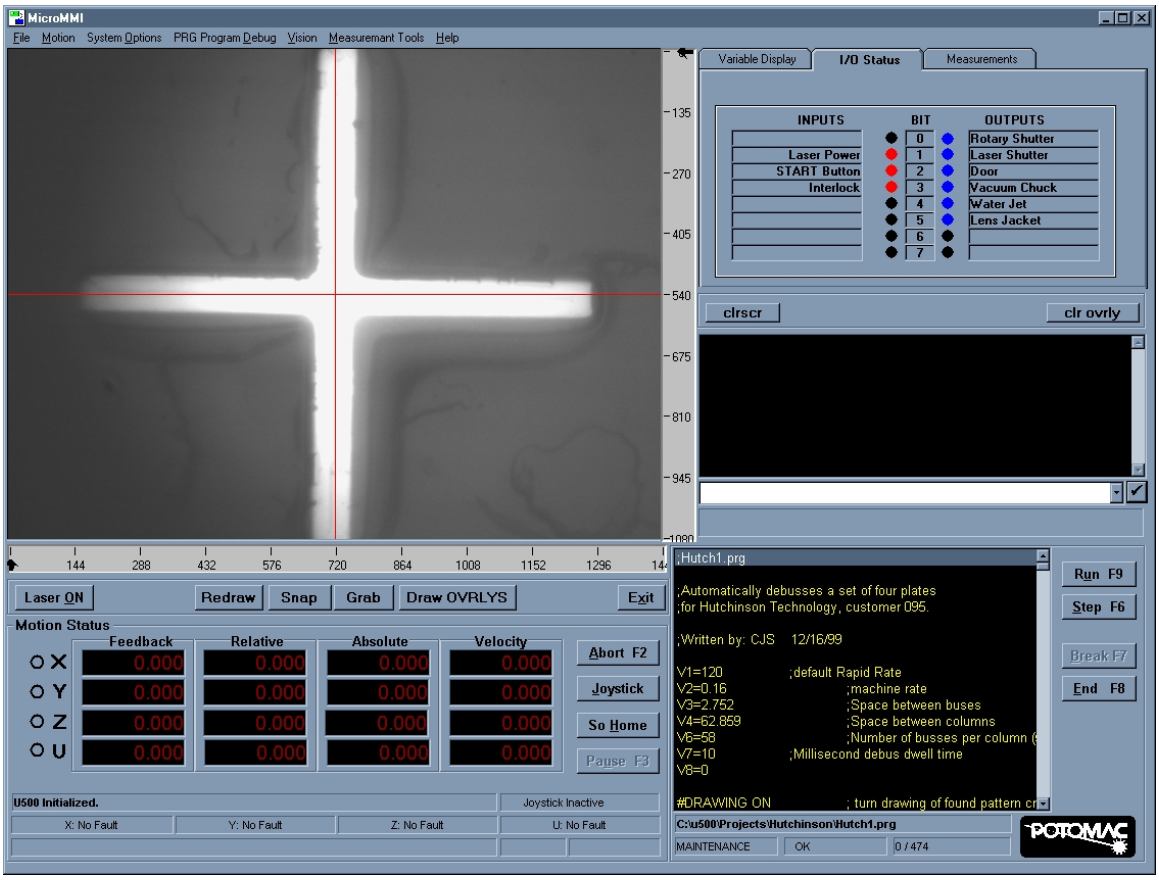
MicroMMI provides the above functionality but it also provides extended debugging functionality. Typical debuggers provide the following functions:

a) **Run** - runs the program from the first line of executable code until a break condition is met or until the end of the program is found
b) **Step** - steps through the program by executing one line of code per each selection of the step command (Step button or Step menu item)
c) **Break** - stops the execution of the program when it is running at whichever line of code the execution was on when the Break command was issued - allows the program to continue executing from the place where it stopped by the selection of the Run command (or continue command depending on the IDE interface)
d) **End** - prematurely stops the execution of the program and resets the program, thereby not allowing the user to continue execution like the Break command

This means that the IDE will be in one of the following states: EDITING, RUNNING, BREAK, or IDLE. When the MicroMMI application is first started the application is in the IDLE state. After the Run button is selected the program is in the RUNNING state. When the Break button is selected the application is in the BREAK state. After the PRG program execution is finished either by reaching the EXit command in the PRG program or after the user selects the End button, the application is in the IDLE state. Any time the user opens the editor and makes a change to the PRG source code, the MicroMMI application is in the EDITING state. The application will remain in that EDITING state until the PRG program is saved, when the application will revert back to the state that the application was in before it was edited.

3 MicroMMI Interface

MicroMMI consists of an interface to a micromachining controller, currently the Aerotech UNIDEX 500 series, an interface to a frame grabber, and an editor, which allows the more advanced operator to execute programs written specifically for the UNIDEX 500. The screen that comes up after the operator has run the program looks as follows:



The live video display area appears in the upper left corner. During machining, the video will be displayed allowing the operator to better control the process depending on the results of machining commands executing on the UNIDEX 500 motion controller.

Directly below the live video display is the axis position display area. The axis position display is where the positions of the axes are reported throughout any movement commanded by the motion controller. To the right of the axis position display are several buttons that allow the operator to perform an abort, pause, software home, and use the joystick on the motion system. Below the axis display is a status bar, which provides continuous status update during program execution and other operator functions.

To the right of the axis display area there is a PRG program display area. After the operator has loaded a program, the program's lines of code will be displayed in the program display box. The buttons to the right of the program display box allow the

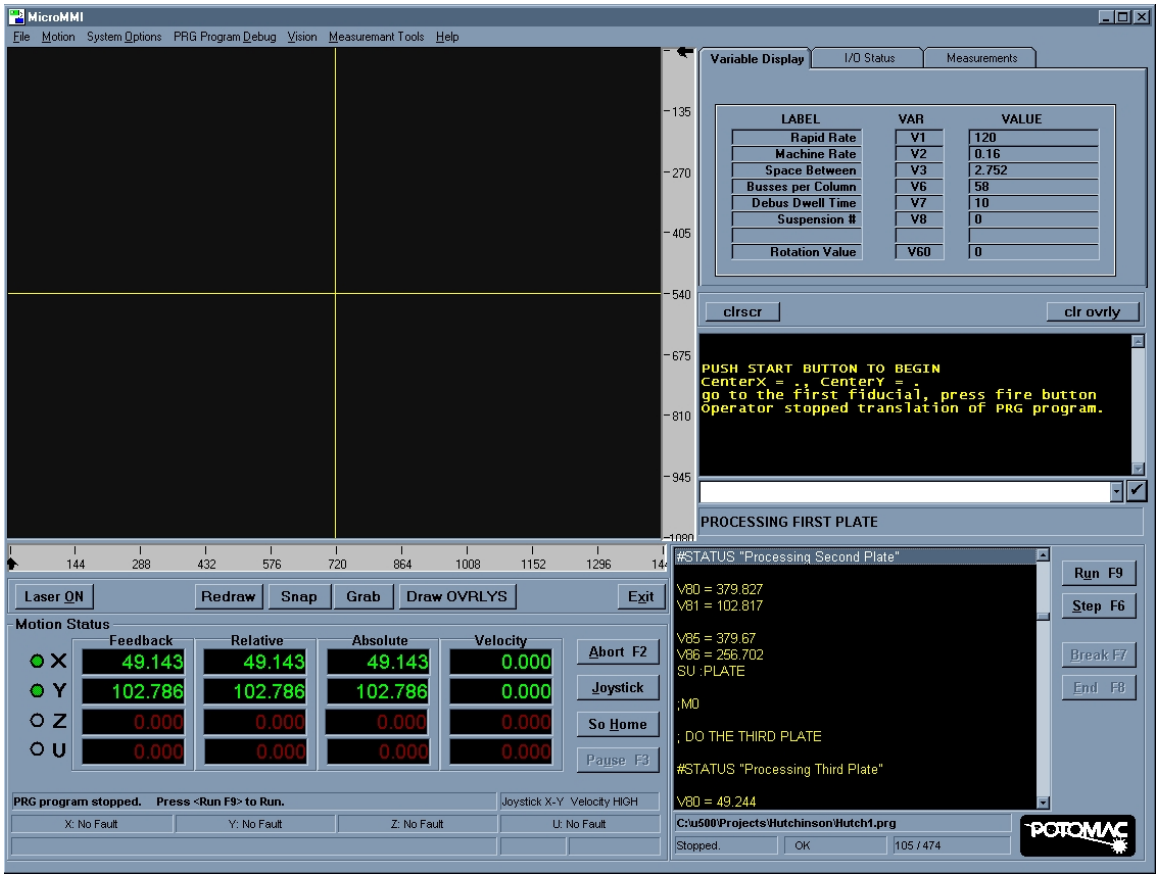
operator to start the program execution, single step through the program, end the program execution, and restart the program. There is a status line available to the U500 PRG programmer via the instruction (#STATUS "Text To Display").

Single U500 commands may be sent to the motion controller using the text entry box above the operator status line. The large black box to the right of the live video display is the operator message box. All messages that are displayed via the U500 message display (ME DI) command are printed to this box.

The tabbed dialog box contains several different groups of data that would be of benefit to the operator at different times.

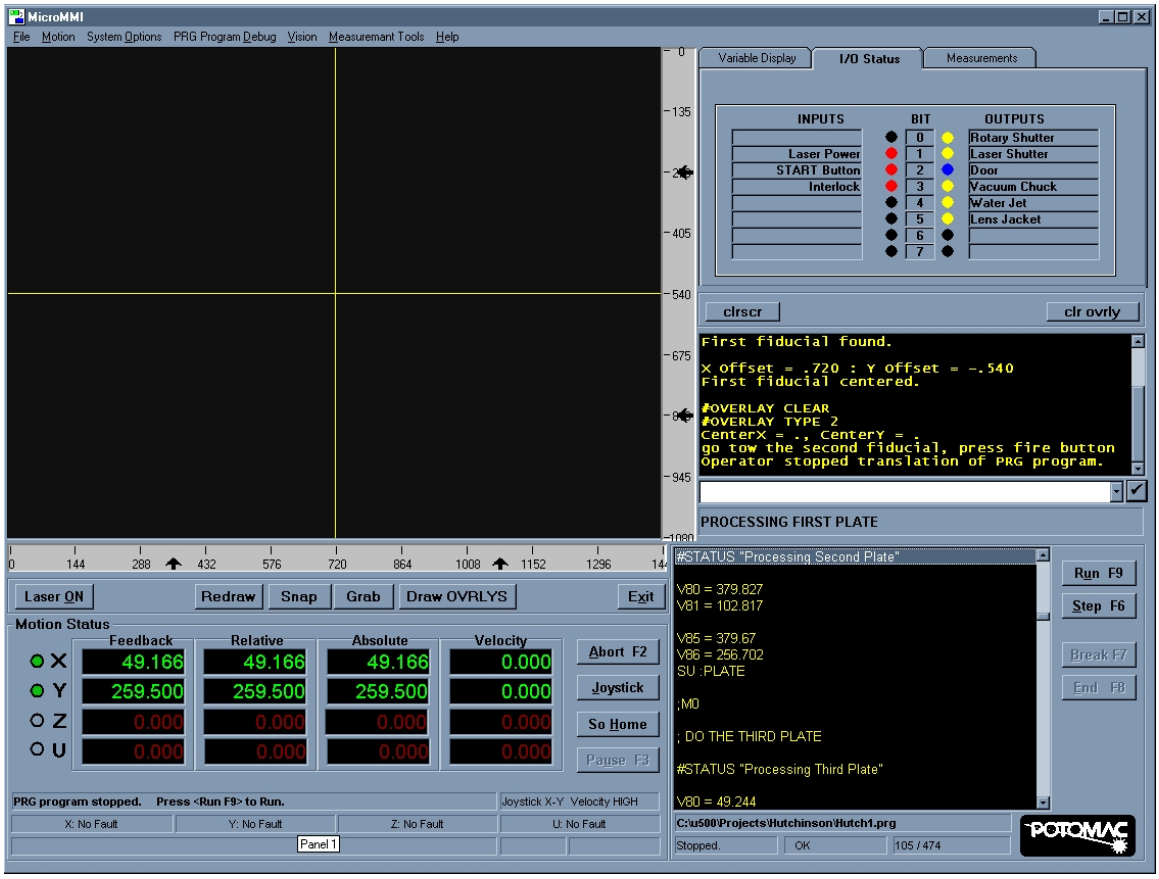
3.1 Variable Display Tab

The variable display allows the operator to select which motion system variables are continuously displayed during PRG program execution. Via the <Watch Variables> menu selection under the <System Options> menu, up to ten variables may be specified for continuous update during processing. A label may be assigned to a variable to better indicate its functionality.



3.2 IO Display Tab

The IO status tab displays the current status of the motion controller's digital Input/Output bits. Via the <IO Labels> menu selection under the <System Options> menu, up to eight inputs and eight outputs may be specified for continuous update during processing. A label may be assigned to the IO bit to better indicate its functionality.

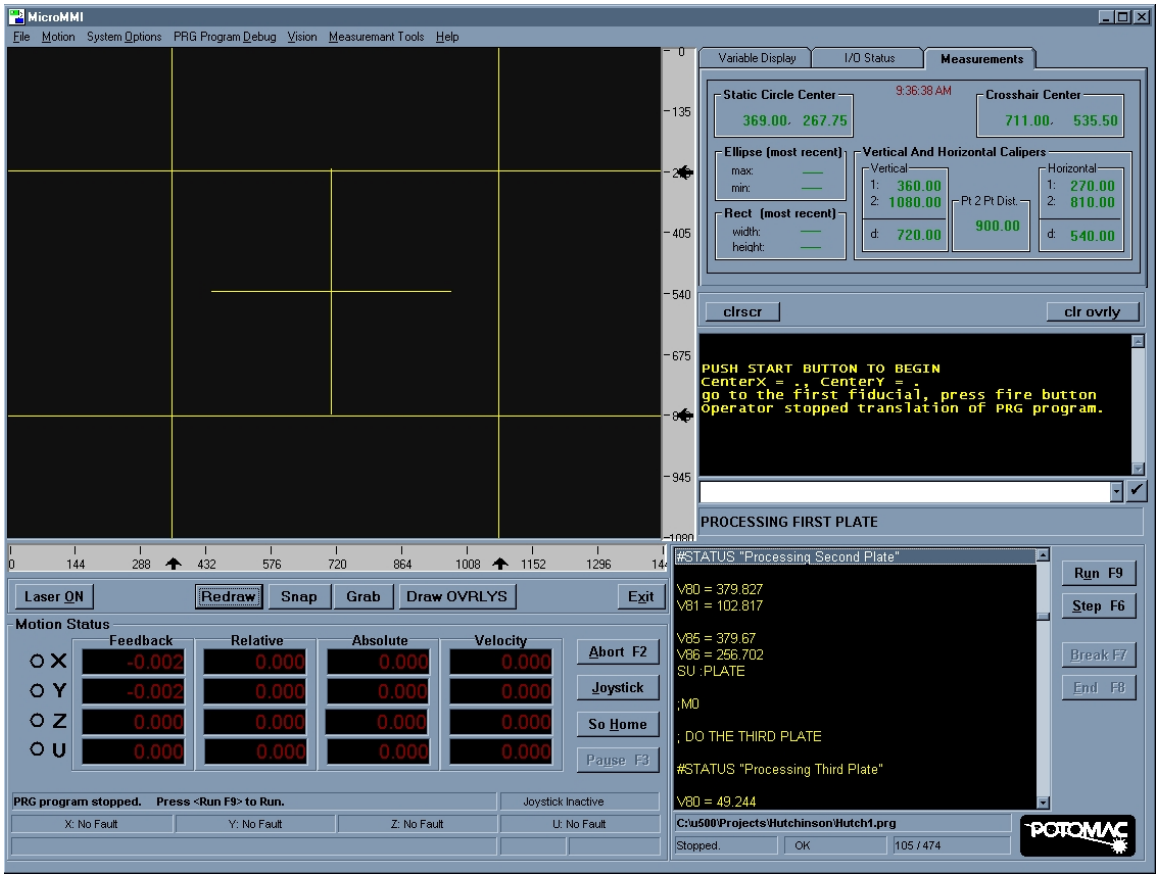


The different colors indicate the enabled/disabled status of the IO bits as specified by the operator via the IO Status dialog box, which appears after selecting the <IO Labels> menu item.

3.3 Measurement Display Tab

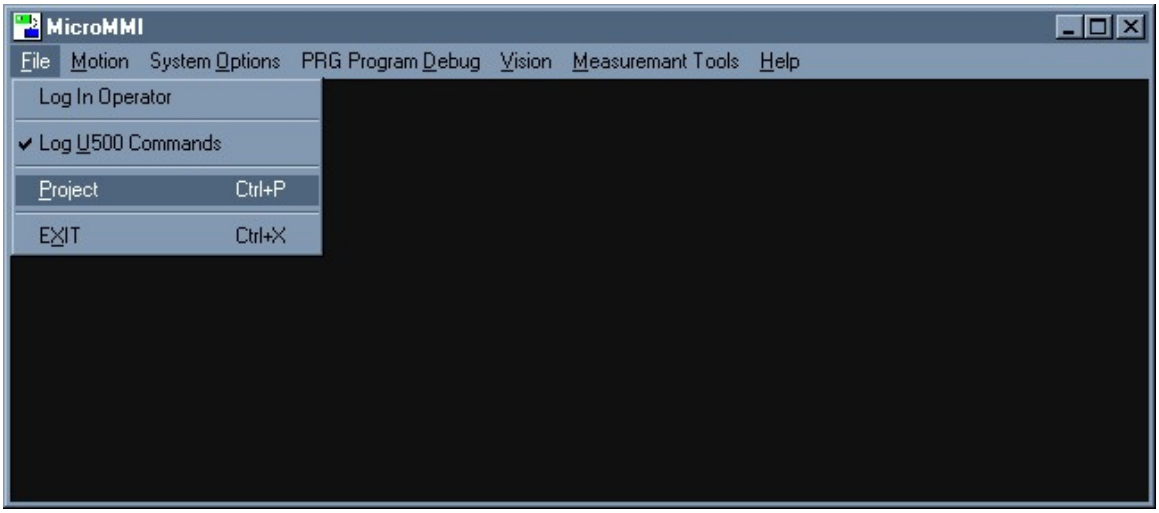
The Measurement Display tab displays the current status of any measurement tools that operator has enabled. Via the <Measurement Tools> menu selection, up to seven separate tools may be enabled and moved about the live video display area to measure part or process features.

The operator has the ability to insert more than one of the ellipse and rectangle measurement tools. Only the last ellipse (rectangle) that was drawn to the live video display will have its positional data updated in the Measurements Tab.


MicroMMI
File Motion System Options PRG Program Debug Vision Measurement Tools Help
Variable Display
I/O Status
Measurements
Static Circle Center
369.00, 267.75
9:36:38 AM
Crosshair Center
711.00, 535.50
Ellipse (most recent)
max:
min:
Vertical And Horizontal Calipers
Vertical
1: 360.00
2: 1080.00
d: 720.00
Pt 2 Pt Dist.
900.00
Horizontal
1: 270.00
2: 810.00
d: 540.00
Rect (most recent)
width:
height:
clrscr
clr ovrly
PUSH START BUTTON TO BEGIN
CenterX = , CenterY =
go to the first fiducial, press fire button
Operator stopped translation of PRG program.
PROCESSING FIRST PLATE
Laser ON
Redraw
Snap
Grab
Draw OVRLYS
Exit
Motion Status
Feedback
Relative
Absolute
Velocity
X -0.002 0.000 0.000 0.000
Y -0.002 0.000 0.000 0.000
Z 0.000 0.000 0.000 0.000
U 0.000 0.000 0.000 0.000
Abort F2
Joystick
So Home
Pause F3
#STATUS "Processing Second Plate"
V80 = 379.827
V81 = 102.817
V85 = 379.67
V86 = 256.702
SU :PLATE
;M0
; DO THE THIRD PLATE
#STATUS "Processing Third Plate"
V80 = 49.244
Run F9
Step F6
Break F7
End F8
PRG program stopped. Press <Run F9> to Run.
Joystick Inactive
X: No Fault
Y: No Fault
Z: No Fault
U: No Fault
C:\u500\Projects\Hutchinson\Hutch1.prg
Stopped.
OK
105 / 474
POTOMAC

4 File Menu Selection

General program functionality is available through the File menu selection.

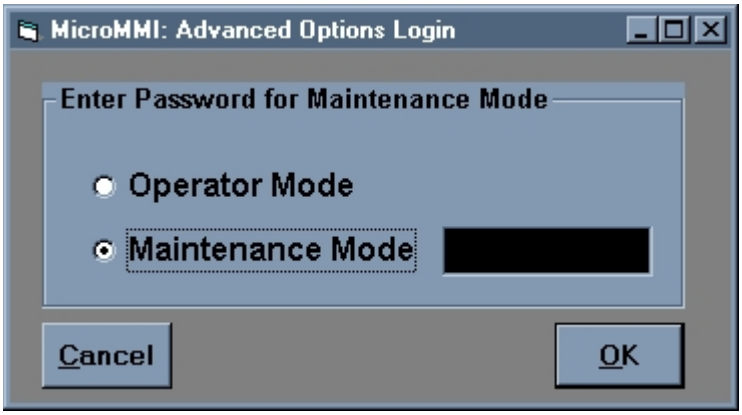


4.1 Log In Operator

MicroMMI has been designed for use in the production environment. In order ensure proper part and system operation, MicroMMI limits the user's access to those functions that are required to complete the production operation. The first time the program is run, the level of access to the program functionality will be at the minimum. The minimum access level is referred to as Operator Mode. During Operator Mode, the only functions that will be available will be the following:

Run Program Button
End Program Button
Abort Program Button
Security Button
Exit Button

In order to activate the rest of the program functionality, the operator must click on the Security Button (under the live video display) and the Security Options screen will pop up. This screen provides two options, Operator Mode, and Maintenance Mode.



If at any time the Maintenance Mode selection is made without entering the appropriate password, and the OK button is pressed, the maintenance functionality will be disabled and the Operator Mode will be enabled. The correct password must be typed into the edit area in order for the maintenance functionality to become available.

The MicroMMI application runs under the Windows NT operating system which m allows multiple users to log into the computer and establish their own working environment via 'profiles'. Potomac installs all applications in the Administrator profile. Therefore, if a user is added to the computer a new profile will be created for that user. At the time of the creation of the new profile, the administrator's profile must be copied to the new user's profile so that the new user will have access to all of the applications that are available to the administrator. It also means that any MicroMMI (or other Potomac application) setup parameter values (such as System Options) must be reset the first time the user runs the specific application.

NOTE: When creating a new user account under Windows NT, copy down all the values of the application's parameters used in the administrator account, copy the administrator account to the new user, and use the copied administrator MicroMMI system parameter values to change the defaults that come up the first time the new user runs the application. Also, the operational mode will have to be reset to Maintenance by entering the Maintenance Mode password again.

The password has been set by Potomac and is provided when the system is delivered. The password is not displayed as it is typed. Instead, asterisks will be displayed in the place of each character typed. If an incorrect character is typed, use the backspace key to erase the incorrect character. Do not use the mouse to select the incorrect characters.

NOTE: Do not use the mouse to select and clear the password or an incorrect password selection, as that will cause the security screen to not function properly. If the correct password has been typed and the program does not recognize it, close the Security screen and re-open it using the Security button. Then type the correct password and the software will recognize it.

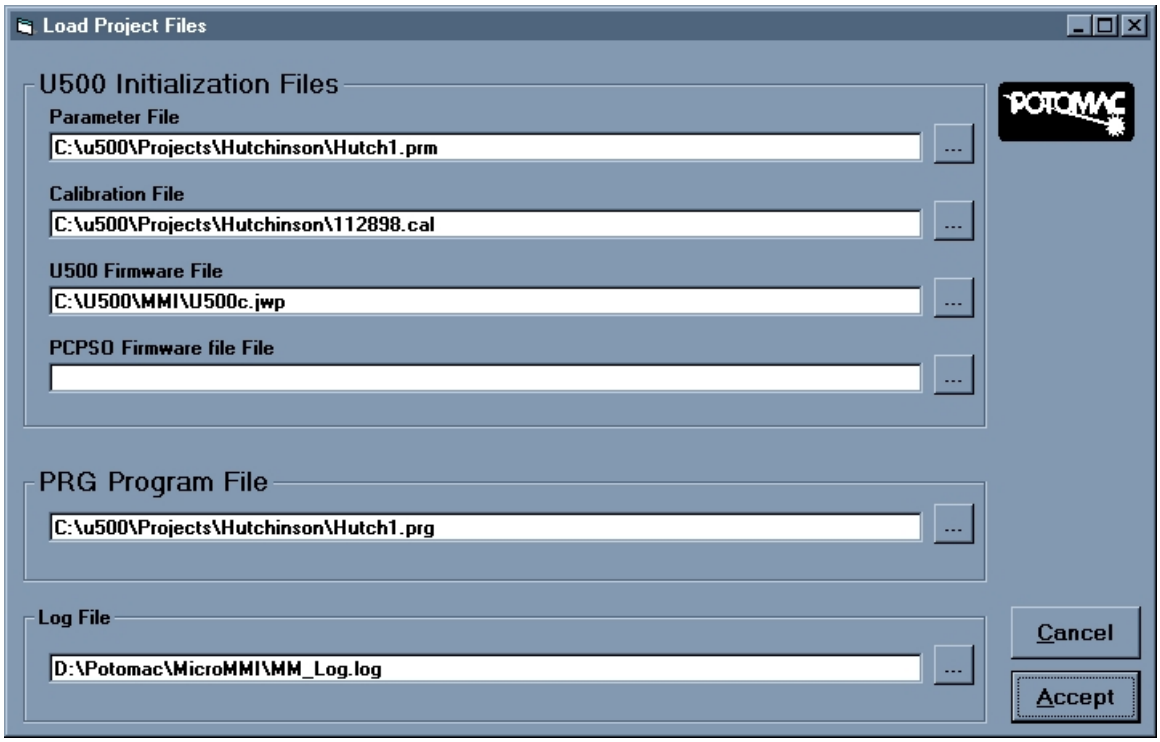
4.2 Log U500 Commands

Whether or not this file is written depends on whether the operator has selected the <File> <Log U500 Commands> menu selection. When the operator selects this, then all of the commands that follow are written to the log file. The user specifies the name of the log file via the dialog box that opens when the user chooses <Load Project Files> menu item. The log file is overwritten each time MicroMMI is run so make sure to copy it to another file name if it is desired to save it. This MicroMMI attribute is not saved from one instance of MicroMMI to the next.

4.3 Project

MicroMMI is software that allows an operator to write a PRG program, load it into the MicroMMI interface, and then execute that program on the LMT micromachining system. As long as the PRG program that has been loaded follows the syntax specified

by the Aerotech MMI PRG program specification and the Potomac proprietary format, the program should execute just fine from within MicroMMI.



The Load Project Files screen that is displayed after the <File> <Project> menu selection has been chosen is where all of the parameter and configuration files that MicroMMI uses are specified. After a valid program name is entered on any of these lines and the Accept button is pressed, the file selections will be saved in the PCs registry. Therefore, the selections will be saved from run to run. **NOTE: Be sure to reinitialize the motion controller after a new parameter file has been chosen.**

4.3.1 Parameter File

The parameter file corresponds to the parameter file used for the Aerotech motion controller. **Make backup and don't modify this file unless you really know what you are doing. Incorrect parameters can damage your motion system equipment.**

4.3.2 Firmware File

Potomac also specifies this file when the system is shipped.

4.3.3 PSO File

Specified by Potomac if this option is enabled

NOTE: The above three files are specified when Potomac ships a system and should NEVER be changed unless directed to do so by Potomac personnel. Always include these files in any backups that are made of the system. The motion controller will not run properly without these files having the correct data in them.

4.3.4 PRG Program File

This is where the PRG motion control program file is specified. This is a file type that can be changed by the operator, but only when in maintenance mode. The PRG motion control file cannot be changed unless the maintenance mode is enabled. The file name that is loaded here will be the file that is displayed when MicroMMI starts. If this file is deleted, either no file will be loaded, or a file that has the same name and resides in the MicroMMI application directory, or another within the system PATH specification, will be loaded instead.

NOTE: Be very careful of having many files around with the same name and containing different data. MicroMMI will always default to the directory specified in the PRG Program File path first, and then it will look through the PATH specification in the Windows NT environment when it cannot find a file. So make sure that multiple copies of files have not been saved there by accident.

4.3.5 Log Files

This file contains a copy of all of the commands that have been sent to the U500. Whether or not this file is written to will depend on whether the operator has selected the <File> <Log U500 Commands> menu selection. When the operator selects this, then all of the commands that follow are written to the log file.

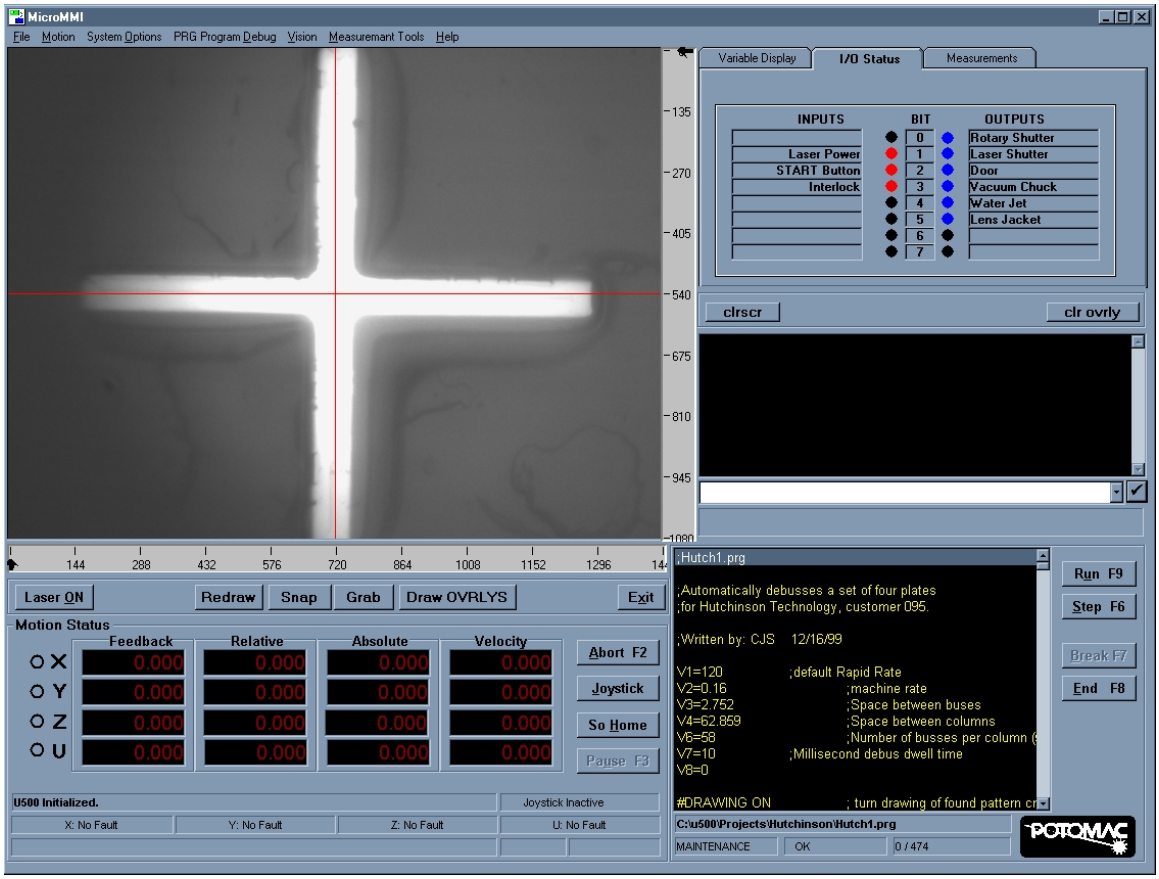
The log file is overwritten each time MicroMMI is run so make sure to copy it to another file name if it must be saved. The log file name is not saved from one instance of MicroMMI to the next.

5 Live Video Display Area

Live video is displayed in the live video display area throughout the duration of program execution. The live video comes from a frame grabber that is attached to an analog video camera of any variety that can make the physical connection. This includes CCD cameras, camcorders, and digital cameras that have analog output.

The live video display is measured in pixels. The dimensions of the video picture are dependent upon the camera hardware as the camera specifies the number of horizontal lines (HSYNCH) it is capable of capturing (MicroMMI usually ships with a 640x480 pixel image). The origin of the picture is in the upper left corner of the display and cannot be changed.

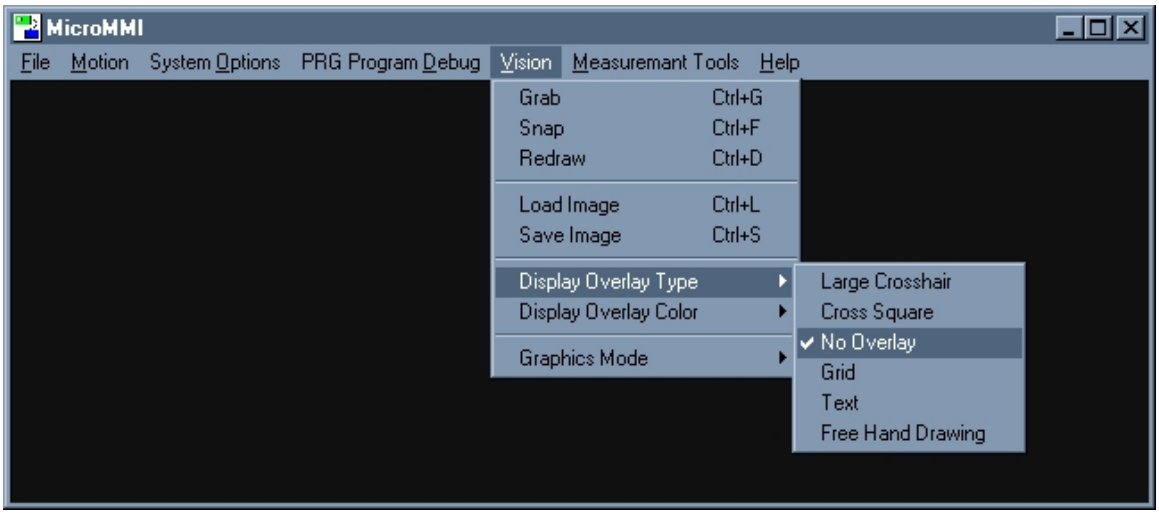
During a production run, the live video display is usually pointed toward the part that is being processed. The status of the executing program can be monitored via this video display. When programs are executed that contain Potomac proprietary vision commands, then the machine vision aspects of the program are being utilized.



Potomac is currently experimenting with several types of machine vision solutions to augment the machining process. Our goal is to increase the speed, compensate for the variability when possible, and improve the reliability of the overall machining process.

This may be achieved by reducing the number of times that the operator needs to interact with the system, thereby decreasing process duration and increasing process throughput. If more inspection work can be allocated to the PC and less to the operator, the PC can usually achieve a much higher production throughput.

6 Vision Menu



6.1.1 Grab, Snap, and Redraw Menu Items

The <Grab> and <Snap> buttons allow the operator to grab live video continuously and to turn live video off by snapping a single image.

Live video is generally used when the part processing must be viewed in order to determine processing acceptability. The <Snap> button may be selected to turn the live video acquisition off thereby allowing for the execution of a PRG program to continue at an extremely high rate.

The <Redraw> menu item is typically used to restore the palette of the live video display area when another window has been displayed on top of the live video. As the MicroMMI application executes in 256-color mode (see Display Properties), the palette of a displayed grayscale image will often times be corrupted by another window that changes the operating system display palette. To restore the palette, select the <Redraw> menu item or press the Redraw button on the button bar below the live video display.

NOTE: When selecting the <Grab> button has enabled live video mode, the performance of the system when running a PRG part program will be significantly affected. Live video requires the real-time transfer a 640x480x8 bit image to the display and will take up processing time away from the PRG program execution. For best performance, be sure to close ALL OTHER applications when running MicroMMI. Also, when minimizing MicroMMI. Be sure to press the <Snap> button so that the live video data is not being transferred to a display that is not being viewed.

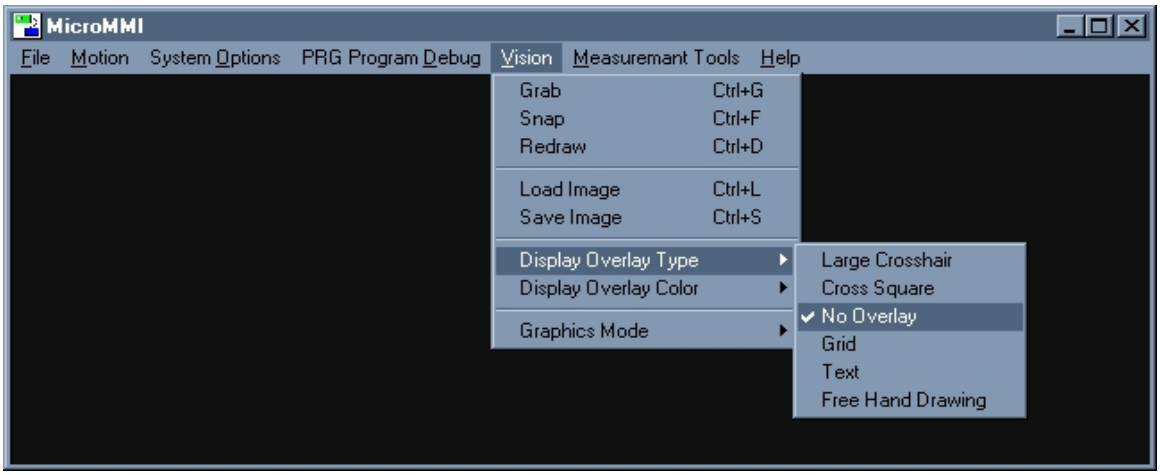
6.1.2 Load and Save Image Menu Items

The <Load Image> and <Save Image> buttons allow the operator to save a copy of the current video display area and to reload it at a later date. This functionality is extremely

useful when training a Region of Interest to be used as a Search pattern at a later date. The image file types that are available are the following: BMP, TIFF, PCX, ITI, and Cognex. (Use Microsoft Photo Editor for conversion to and from available image formats.)

6.1.3 Overlay Type Menu Item

Selecting the <Display Overlay Type> menu item will provide for the change the overlay type. There are many overlay graphics from which to choose. The graphics comprise two categories: static and dynamic. The static overlays are those that once they are placed on the display area, they do not move and cannot be redrawn by the operator. Static overlays are generally used as a reference to a known position such as the center of the field of view. The dynamic overlays may be selected and then multiple instances of each type of graphic may be drawn to the display area as many times as is desired. To draw the dynamic graphics, select the type of graphic to be drawn and then click on the display area where the graphic is to be started. Drag the mouse in the desired direction and the graphic should shrink and expand accordingly.



6.1.4 Overlay Color Menu Item

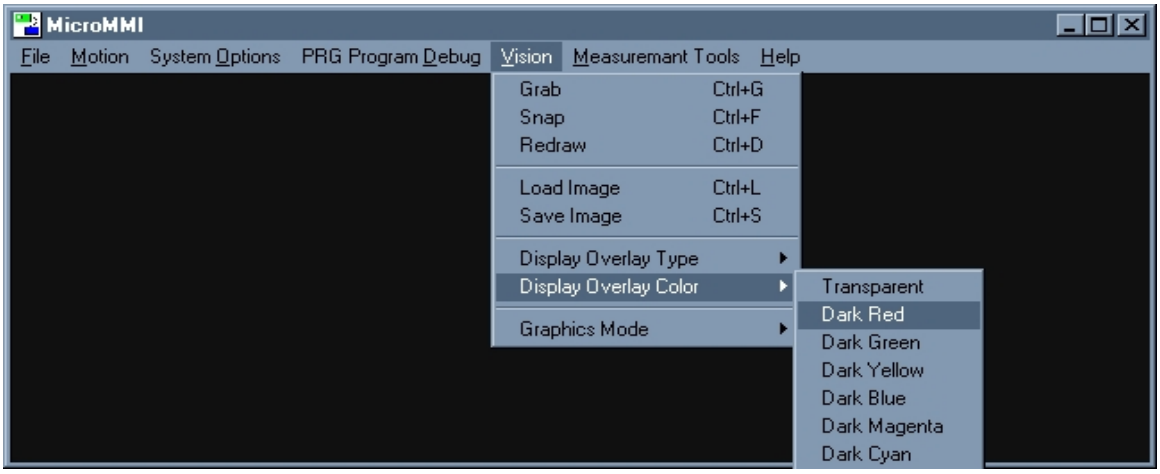
The overlay graphic colors may be selected using the <Display Overlay Color> menu item. The color of the overlay graphics may be changed however the line thickness may not be changed at this time.

6.1.5 Graphics Mode Menu Item

The graphics mode indicates the hardware and software that is used to generate the live video and the overlay graphics to the live video display. All methods of transfer of the camera data from the frame grabber to the live video display have their advantages and disadvantages. The DIB_IMAGE menu item will enable the most flexible method of transferring live video but it is also the slowest methodology. Each frame of camera data will pass through the computer's RAM before it is ultimately delivered to the rightful party. The DDRAW mode is considered a hardware transfer because it uses the Microsoft DirectX functionality to transfer the camera data directly from the frame grabber memory straight to the live video adapter memory (and therefore directly to the screen display.) DDRAW is a much faster method of camera data transfer but it requires

the graphics adapter to have full overlay capability and limits the number of colors used to 256.

NOTE: Potomac currently delivers an ATI Rage Pro graphics adapter with a slightly out of date adapter driver because functionality that ATI correctly implemented in their driver no longer works in their most recently updated driver.



6.1.6 Camera Menu Item

Up to four cameras may be connected to the system (depending on the type of frame grabber delivered with the system). The camera input connectors are labeled on the camera cable if the appropriate cable has been purchased. If the cable does not allow for four cameras, a new cable may be purchased from Potomac. The camera selection is made through the <Camera> menu item. The default will always be camera 1.

NOTE: NEVER, NEVER plug the camera into the frame grabber board while the computer is powered up. The camera provides signals to the frame grabber board that indicate the resolution and "hot" plugging the cable into the frame grabber may (and will eventually) destroy the frame grabber hardware and the other devices in the computer.

7 Video Button Bar

The button bar below the live video display area contains buttons that control some aspects of the machine vision and some which are used generally throughout the operation of the program. Any of these buttons may also be called from within a PRG program by using the appropriate syntax for the command.

For example: a call to the camera functionality would be the following:

#SNAP
#GRAB

And so on. The Draw OVRLYS button does not have an equivalent PRG program syntax and is only used to switch between the drawing of overlays to the live video display and the drawing of Regions Of Interest (ROIs).

7.1 Laser ON Button

The Laser On button is used to toggle the laser on and off. This button will toggle between on and off depending on the state of the laser.

7.2 Redraw Button

The Redraw button will redisplay the live video display area including any *static* graphical overlays that have been selected by the operator. The static graphical overlays are the LARGE CROSSHAIRS, TEXT, GRID and the CROSS SQUARE. These overlays do not have to be drawn by the operator as with the dynamic graphical overlays such as ELLIPSE, RECTANGLE and FREE HAND DRAWING.

The Redraw button is also used whenever the live video display area has lost its palette. Because the MicroMMI application runs in 256-color mode (see Display Properties), the display area palette may be lost each time a window other than the display area, receives the focus or is drawn. Selecting the Redraw button will correct the multicolor symptoms of a lost palette.

7.3 Snap Button

The Snap button is used to display only a single frame of video data from the camera to the live video display area.

7.4 Grab Button

The Grab button will display live video data continuously to the live video display area until another button is pressed or a command has been executed from within a PRG program that halts the live video display.

7.5 Draw OVRLYS Button

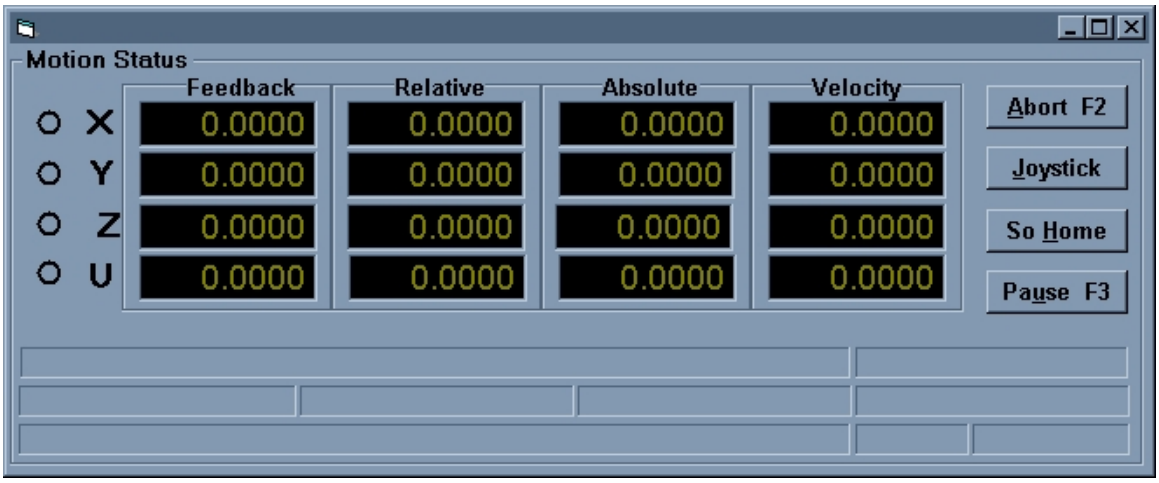
MicroMMI contains machine vision functionality that is implemented via the PRG programming capability. It has been determined that ability to create Regions Of Interest directly within the MicroMMI application is a necessary capability. Therefore when this button says Draw ROIs the user may create an ROI directly within the live video display area by clicking with the left mouse button where the upper left hand corner of the ROI should be and dragging until the lower right hand corner is placed in the desired location. A green bounding box will show where the bounds of the rectangular ROI exist and a dialog box will appear prompting for the name of the newly created ROI.

7.6 Exit Button

The Exit button does just what it sounds like, it ends the operation of the program.

8 Axis Position Display

The MicroMMI program has been written to have direct communication with the UNIDEX U500 motion controller. The motion axes status and position values that are controlled via PRG programming commends are always be available. The axis position display area is a standard display which shows the relative, absolute, and feedback positions of the axes during any given program execution.



The axis positions are also displayed when the operator is using the joystick in order to move around.

The axis display area provides a button bar that allows the operator to abort a move, perform a software home, pause a series of commands, or enable the joystick mode. To set the values of the position registers back to zero, the motion controller can either be reset or software home can be performed.

NOTE: The axes must be enabled in order for the joystick to work as expected.

NOTE: Be sure to deactivate the JOYSTICK before running a PRG program because the U500 controller does not process any commands sent to it when it is in JOYSTICK (or SLEW) mode.

9 PRG Program Display

The program display area is used to display the motion control program that has been selected by the operator. The operator may select a motion control program either via the <File> <Project> or the <Motion> <Load Motion Program> menu selections. The <File> <Project> screen will save the loaded program file name from one instance of MicroMMI to the next.

MicroMMI provides extended debugging functionality. Typical debuggers provide the following functions:

a) **Run** - runs the program from the first line of executable code until a break condition is met or until the end of the program is found

b) **Step** - steps through the program by executing one line of code per each selection of the step command (Step button or Step menu item)

c) **Break** - stops the execution of the program when it is running at whichever line of code the execution was on when the Break command was issued - allows the program to continue executing from the place where it stopped by the selection of the Run command (or continue command depending on the IDE interface)

d) **End** - prematurely stops the execution of the program and resets the program, thereby not allowing the user to continue execution like the Break command

This means that the IDE will be in one of the following states: EDITING, RUNNING, BREAK, or IDLE. When the MicroMMI application is first started the application is in the IDLE state. After the Run button is selected the program is in the RUNNING state. When the Break button is selected the application is in the BREAK state. After the PRG program execution is finished either by reaching the EXit command in the PRG program or after the user selects the End button, the application is in the IDLE state. Any time the user opens the editor and makes a change to the PRG source code, the MicroMMI application is in the EDITING state. The application will remain in that EDITING state until the PRG program is saved, when the application will revert back to the state that the application was in before it was edited.

9.1 Running PRG Programs

The loaded PRG program can be executed by pressing the Run button on the right side of the PRG program display area or by pressing F9. The loaded program will execute all valid MicroMMI instructions. Valid MicroMMI instructions are specified in Appendix A. The Start button is available in all operating modes.

9.2 Ending a PRG Program

The program display will scroll along with the program's execution so that the operator may view the exact lines being executed. The End button can be pressed at any time during the execution to completely stop the program. When the End button is pressed, the program will restart from the top (or beginning) of the PRG program. The End button is available in all operating modes.

9.3 Stepping a PRG Program

A PRG program can be stepped in order to debug that program. For each Step button press, a single line of PRG code will be executed. The Run button can be selected at any time during a stepping operation. If the Run button is selected, it will operate similar to a 'continue' operation in that the normal execution of the program will continue until the end of the program is reached or the operator selects the End button. The Step button is available only in maintenance mode.

NOTE: If the <Step> command becomes unavailable during program execution, such as when it reaches a LOOP command, SUB command, etc., the <Break> button must be selected in order to continue the stepping operation. Not required for normal operation.

NOTE: The laser should not be enabled in a program that is being debugged because the laser may stay too long in a particular location.

9.4 Breaking a PRG Program

Because the ability to Break into the execution of a program is provided, there is also the ability to continue execution from the point where the program was stopped. MicroMMI uses the Run button to continue execution after any break has been performed. There are several ways to 'break' program execution. They are 1) to press the Break button, 2) set a breakpoint on a specific line of PRG code, and 3) arrive at an 'M0' PRG programming command. After any of these break conditions have been met, the user has several options. The user may 1) select the Run command by pressing the Run button or the Run menu item, 2) select the Step command by pressing the Step button or the Step menu item, or 3) ending the program execution and resetting the program by selecting the End button or the End menu item. Therefore the Run button may be enabled even though the program is in the 'Running' state.

9.5 Restart a PRG Program

Restarting ends program execution and repositions the instruction pointer to the beginning of the program. Program execution begins immediately. To perform a restart, press the End button and then press the Run button. .

9.6 PRG Command Text Entry Box

The operator uses the edit line above the display box to enter any PRG program single line instruction. This edit line will remember previous instructions that were entered. The operator may access previously run commands by clicking on the Check Mark (√) button next to the text entry box. The command entry text box is as follows:

9.7 Using the Command Text Entry Box to Enter Special Keys

The PRG command text entry box can also be used to activate the user defined special function keys. There are only 5 function keys available in this release. If any numbers greater than 5 are used to specify a function key, an error will be reported. The syntax for the function keys corresponds to the Aerotech definition for user defined function keys, but with one addition. The addition is the ability to define a Potomac Proprietary Interface, (PPI), type of function key. The syntax for specifying a PPI function key is as follows:

SKEY PPI	Key #,	PRG Command,	Key Text
SKEY PPI	3,	EN X,	X Axis
SKEY PPI	1,	CLRSCR,	Clear
SKEY PPI	5,	#OVERLAY CLEAR	CLR OVRY
SKEY PPI	4,	SU :TRAIN,	TRAIN

As soon as the SKEY command is entered into the PRG command entry box, the user defined function key is added to the user defined button bar. If the operator had entered the above four commands into the command text entry box, the special keys available would look like the following:

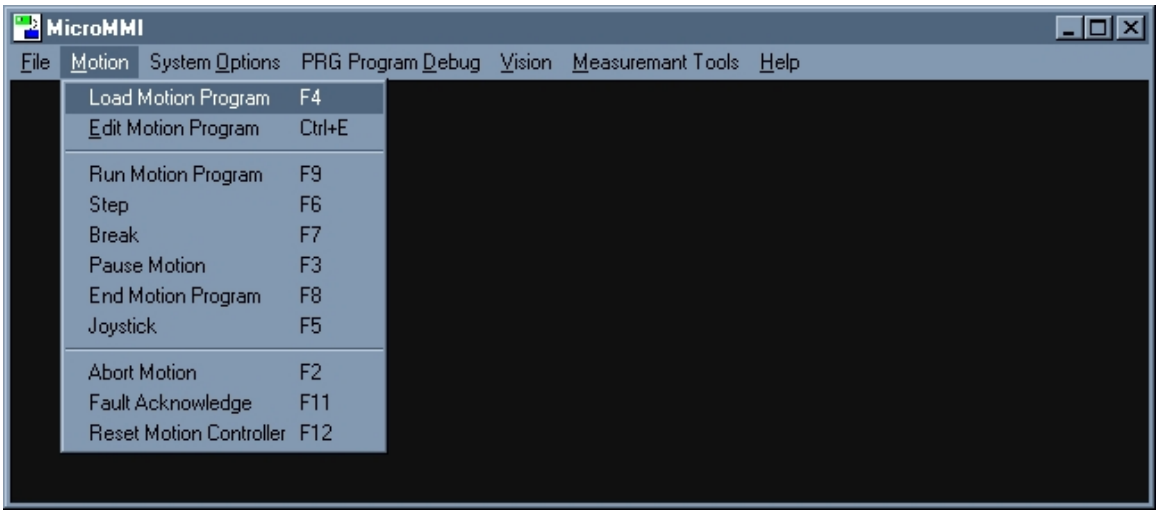
Any keys that have not been enabled do not appear on the special key button bar. After being enabled, any key that is pressed executes whatever PRG command instruction has been assigned to it. For example, pressing the X Axis key causes the message ‘EN X’ to be passed to the PRG command text box for interpretation and execution by the UNIDEX 500 motion controller. The special key functions are only available in maintenance mode.

NOTE: At this time, only the ability to execute SINGLE line commands is implemented in the function keys although a subroutine may be called from a special function key as long as the subroutine is contained in the currently open PRG file.

The special function keys are used for debugging in that a current value of a variable may be recorded to the operator status display after a break point has been reached, or a Pause (M0) command has been reached. If the special key has been programmed to display the value of a variable to the operator message area, the corresponding special key need only be pressed in order to get the current variable results. Otherwise, the entire command, ‘ ME DI “Value for V2 = %V2” would have to be typed into the PRG command text box each time the information is needed.

10 Motion Menu Selection

The <Motion> menu contains all selections that operate on the UNIDEX 500 motion controller.



10.1 Load Motion Program

The Load Motion Program menu selection differs from the Load Project Files screen because the program loaded via this selection will not be remembered from one execution of MicroMMI to another. This selection will display the typical file load dialog box and request the name of the file to load into the program display area. This selection is only available in maintenance mode.

10.2 Edit Motion Program

This menu selection brings up the PRG program editor that is included with the MicroMMI application. From within the editor, the PRG programs can be run and debugged similarly to the functionality that appears on the MicroMMI main screen. The editor is only available in maintenance mode.

The MicroMMI application IDE provides the ability to open the editor while the program is running. This ability has been included to allow viewing of a larger amount of the PRG program while the program is running. It also allows the execution of the PRG program while in the editor so that more intensive PRG program debugging is available. At the time the PRG program editor window is closed, however, any execution is stopped and the file is saved to the hard drive.

10.3 Running PRG Programs

The loaded PRG program can be executed by choosing the Run menu selection or by pressing F9. The loaded program will execute all valid MicroMMI instructions. Valid MicroMMI instructions are specified in Appendix A. The Start button is available in all operating modes.

NOTE: Be sure not to leave the Joystick enabled before running a PRG program. When the joystick is enabled the U500 is in 'SLEW" mode and will ignore all subsequent move commands sent to it via the user interface and via the PRG program.

10.4 Ending a PRG Program

The End menu selection can be chosen at any time during the execution to completely stop the program. When the End selection is pressed, the program must be restarted from the beginning. The End selection is available in all operating modes.

10.5 Stepping a PRG Program

A PRG program can be stepped in order to debug that program. For each Step menu selection press, a single line of PRG code will be executed. The Step selection is available only in maintenance mode. Stepping into subroutines is not allowed at this time. To see inside a subroutine, set a breakpoint inside the subroutine and the program execution will stop there.

NOTE: The laser should not be used in a program that is being stepped because the laser may stay too long in a particular location. There is no code in the MicroMMI application, which prevents the use of the laser during Stepping. The operator should take care of this himself.

10.6 Pausing a PRG Program

An execution may be paused and then resumed by choosing the Pause menu selection. The pause will stop any more PRG program lines from being downloaded to the U500 processor for execution. The U500 will continue to execute any instructions it may have in its buffer, but no new instructions will be executed. In that way, a DWELL command will finish before the pause takes affect. The paused program may be resumed at any time by pressing the Pause button again or selecting the Pause menu selection again. This function is only available in maintenance mode.

10.7 Using the Joystick

The joystick may be activated at any time after selecting the joystick menu selection. This function is only available in maintenance mode.

NOTE: The axes that the joystick will control must be enabled before the joystick function will work.

10.8 Aborting Execution

The execution of a PRG program may be aborted by selecting the abort menu selection. When a PRG program has been aborted, the commands that are in the motion controller's instruction buffer are discarded and no further instructions are sent to the processor. The program must be restarted after an abort has been selected. This function is available in all modes of operation.

10.9 Issuing a Fault Acknowledge

A 'fault acknowledge' must be initiated when there has been a fault in the motion controller. Typical faults include axis hardware limits which can be reached if an axis is

driven too far to the edge of its area of operation. This function is only available in maintenance mode.

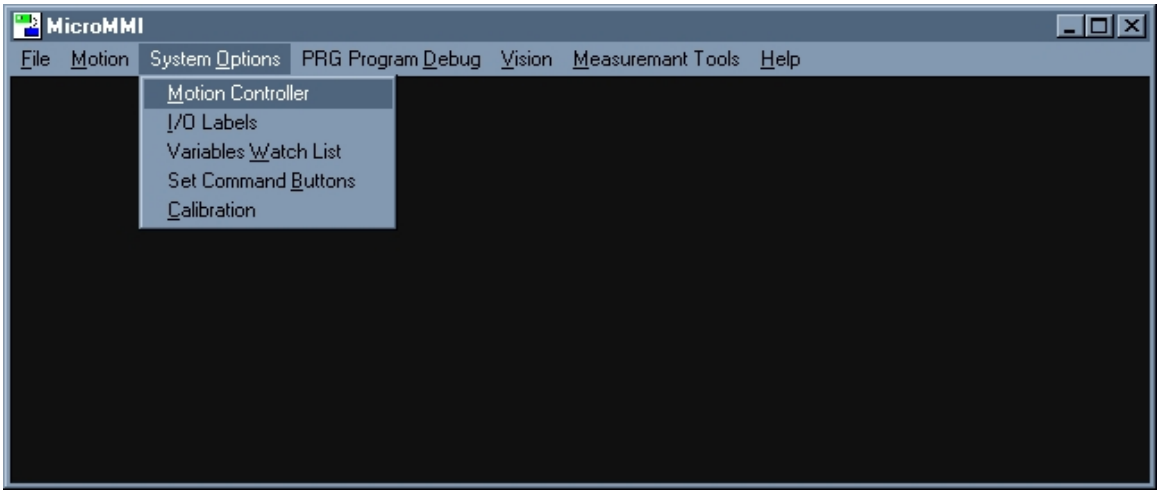
10.10 Resetting the Motion Controller

It is desirable at times to reset the motion controller especially when a new program is about to be run. If there has been a series of faults, resetting the motion controller can initiate a known state at which all variables and stored axis positions are reset to their initial values.

NOTE: The UNIDEX 500 motion controller must be reset when the MicroMMI application is first run.

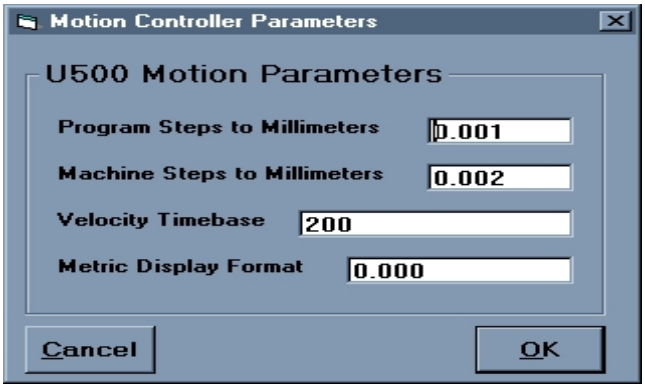
11 System Options Menu

The System Options menu selections allow the operator to change various parameters that determine how the MicroMMI application will run and display data. These parameters are saved to the registry and only need be determined the very first time the application is run. However, if a new user has been added to the computer, that user will have to reload the MicroMMI application variables that were loaded by Potomac upon application delivery. Potomac installs all application code into the Administrator profile. If another user account need be added to the computer, the administrator profile must be copied to the new users profile and these parameters must be reloaded upon first running the application in the new user account.



11.1 Motion Controller Parameters

The UNIDEX 500 motion controller will be shipped from Potomac with several files that must always be used when running the applications that contain motion capability. There are several parameters that may be read from the motion controller parameter file the very first time the MicroMMI application is run. The parameters may or may not be correct. Potomac will set these parameters before the system is delivered, however, these parameters may also be changed at a later date if the application changes significantly from the way it was shipped. Selecting the <Motion Controller> menu item will display a dialog box, which allows the parameters read from the parameter file to be changed.



The only parameter that is not read from the UNIDEX 500 parameter file is the Velocity Time base. This value must be set the first time the application is run.

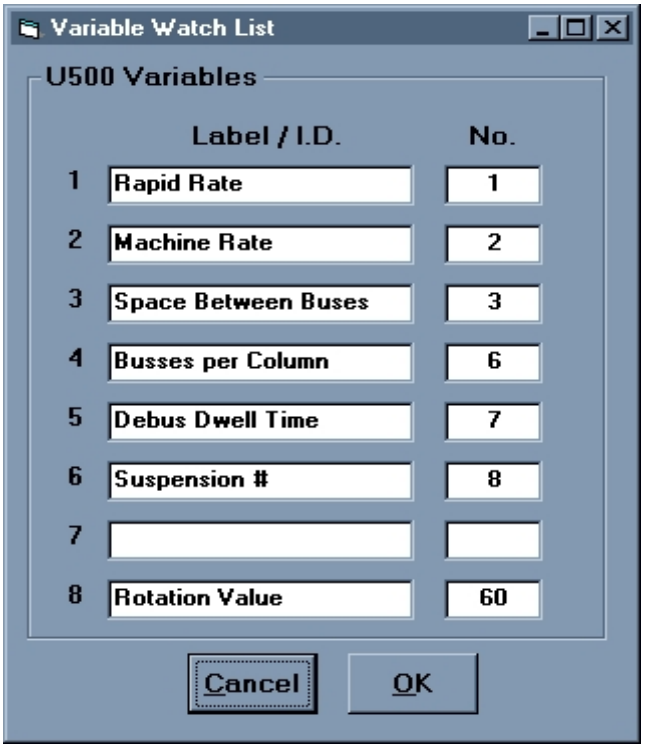
The typical values for the motion controller parameters are the following:

Program Steps to Millimeters:	0.001
Machine Steps to Millimeters:	0.00025
Velocity Timebase:	100
Metric Display Format:	0.000

When MicroMMI is run it will look to the parameter file for the appropriate motion controller values. These values may be then changed via the above dialog box and will remain in effect while the current instance of the application is running. When MicroMMI is closed and then re-run, the values will again be read from the parameter file which is loaded when MicroMMI starts. **NOTE: If the parameter file is changed, and the motion controller values are not known, they may be read from the parameter file by closing the MicroMMI application and re-running it.**

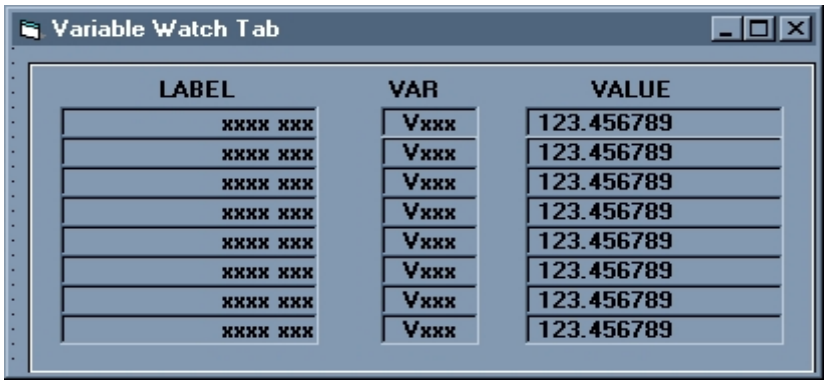
11.2 Variable Watch List

The variables that are watched may be set via the <Variables Watch List> menu selection. The following screen will appear if that menu selection is made:



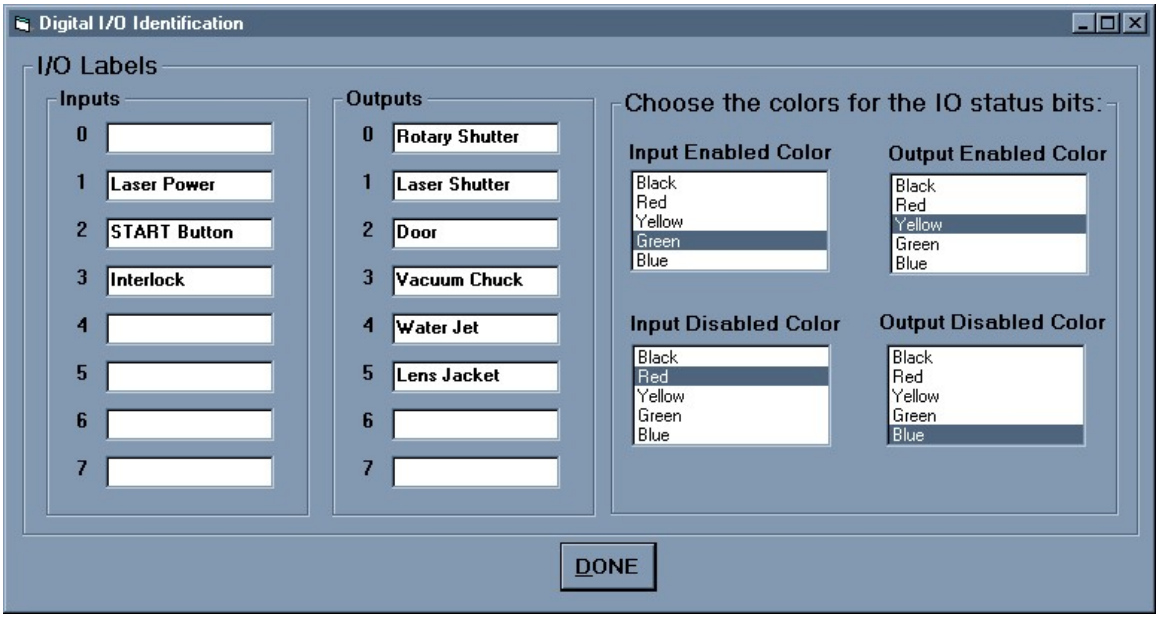
The field called 'No.' corresponds to the motion controller variable number. The operator may assign a symbolic label name to which the variable is referred when it is displayed in the variable display. The variable watch list is only accessible when the program is in maintenance mode.

The variables that are watched at a regular basis can be seen when the Variables Tab on the main MicroMMI application window is selected. The variable values are displayed as follows:



11.3 IO Labels

In order to label the inputs and outputs, the operator must select the <IO Labels> menu selection. The following screen appears when the operator selects the IO Labels menu selection.



The operator may then enter the name for each input and output that has been configured by the hardware. After the names have been entered, they will appear in the IO Status Tab if the operator has selected that tab option on the main MicroMMI application window. The inputs and outputs must be hardwired into the system when it was delivered or set up by qualified technical personnel.

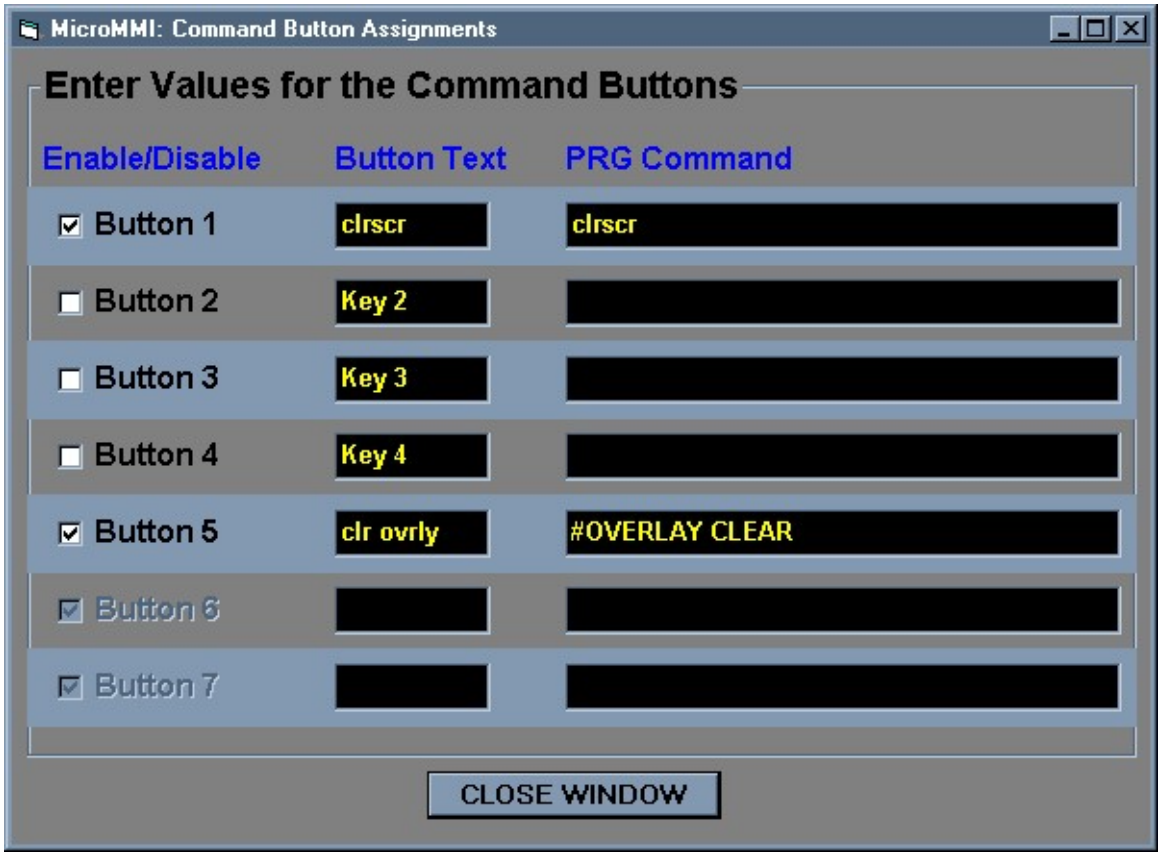
11.4 Set Command Buttons

The <Set Command Buttons> menu selection displays a screen whereby the operator may enter values for a set of user-defined buttons that are enabled on the MicroMMI user interface.

NOTE: These keys do not correspond to the F1 through F5 keys on a computer keyboard. These keys are operator programmable command keys that must be clicked using the left mouse button, in order to be activated. Pressing the F2 key will perform the ABORT command as specified in the user interface NOT perform special key number 2's command functionality.

The function keys are displayed above the operator status area and the PRG program display area on the main MicroMMI application window. After the keys are enabled, they will appear as buttons on the user defined keys as explained in Section 9.7.

The operator can individually program each of these buttons when the program is in maintenance mode. There are two different ways to program the buttons. The first is to select the <Set Command Buttons> menu item from the <System Options> menu. This menu selection will display a screen like the following:



The Button Number is the number of the buttons starting from left to right as displayed on the main application window as seen below. Button 1 corresponds to the first button on the left, button 2 the second button from the left, and so on.

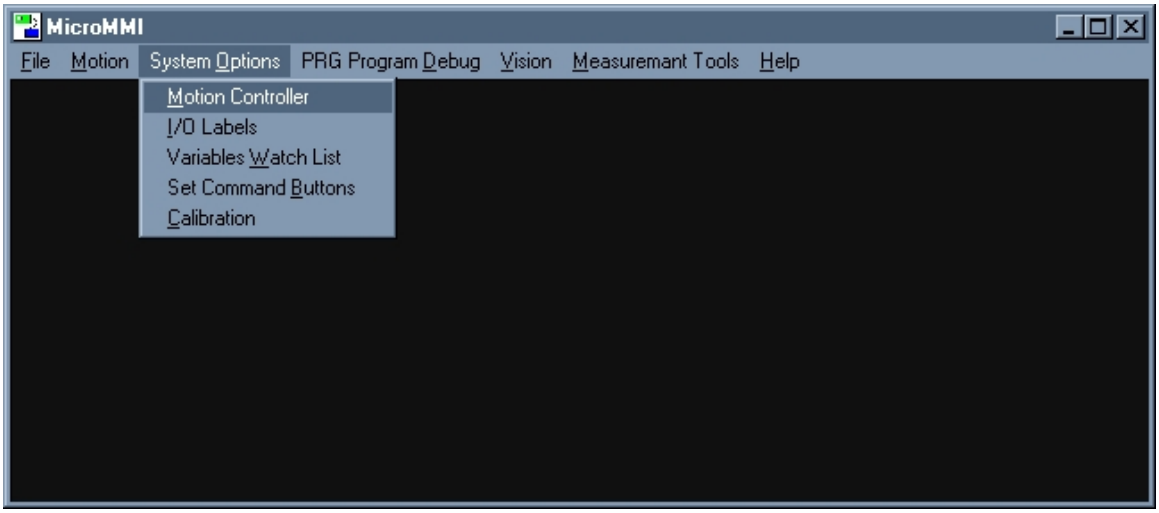
If the box next to the phrase "Button 1' is checked then that button is displayed on the MicroMMI interface, otherwise the data is stored for later use. The button text is the text that should appear on the face of the button when it is displayed in the user defined keys section on the main MicroMMI application interface. The PRG command refers to a single line PRG command, as it would be programmed in a PRG program. When the operator presses a button, MicroMMI executes the corresponding PRG command. At this time, no control structures may be used in the user defined function keys. For example, LOOP 23, can not be used because there is not another line to indicate where to return from the loop.

NOTE: The contents of this display screen are saved in the operating system's Registry and therefore saved from one execution of MicroMMI to the next. If the executable is moved or the program is reinstalled, the values will have to be specified again.

The command buttons are used for debugging in that a current value of a variable may be recorded to the operator status display after a break point has been reached, or a Pause (M0) command has been reached. If the special key has been programmed to display the value of a variable, the operator only needs to press the corresponding button to get the currents variable results.

11.5 Calibration

The <Calibration> menu selection displays a dialog box that allows the operator to enter values that will convert the machine vision pixel space into whatever calibrated units are desired.

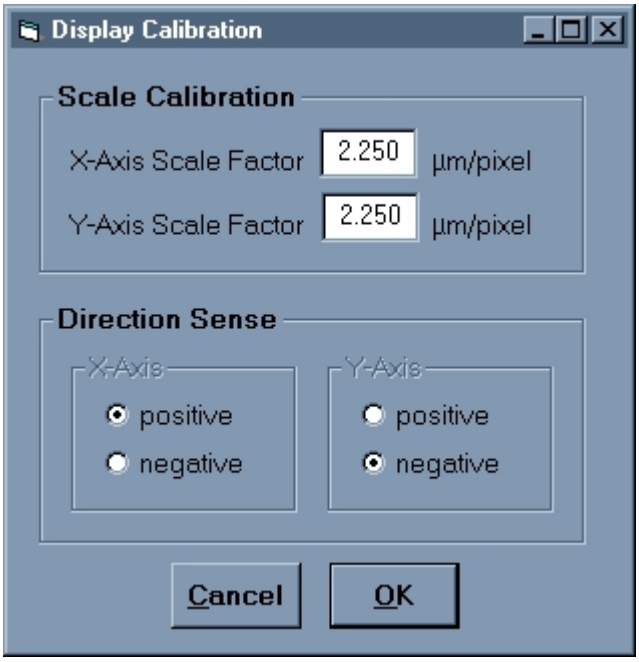


The calibration must be determined by inserting a known object into the field of view so that the live video can completely see it.

1) Using the MicroTools application, select pixels as the desired units in the calibration configuration screen of that application.
2) Then open the Inspector window and click on the Init Video button, then the Acquire Video button. Live video should be in the display area and the known object should be visible.
3) When the known object is completely in view, click on the Freeze Video button.
4) Insert two horizontal (or vertical) measuring calipers into the Frozen video display. (If you do not see them then you probably did not freeze the video by clicking on the Freeze button.)
5) Move the calipers around until each is on either side of the known object so that the distance between the calipers is equivalent to the known object distance.
6) Go to the <Display> - <Tools Configuration> menu item and select the corresponding <Caliper Configuration> menu selection so that the dialog box which allows the display of the caliper results appears. Click the checkbox corresponding to the set of calipers that are being used.
7) Look to the right of the Inspector window and the Delta Results for the calipers that are being used should be highlighted and changing value when the calipers are being moved. This value will correspond to the number of pixels per the known object distance.
8) After you have measured the number of pixels that correspond to the number of known units, the calibration constant can be calculated as follows: cal. Constant = user units / pixels.

For example, if the object is known to be 100 ums and the MicroTools application has been used to measure the number of pixels which was 212, then the calibration constant would be the following: Cal. Constant = 100 / 212 = 0.4716. This would be the value entered into the calibration dialog box in the MicroMMI application.

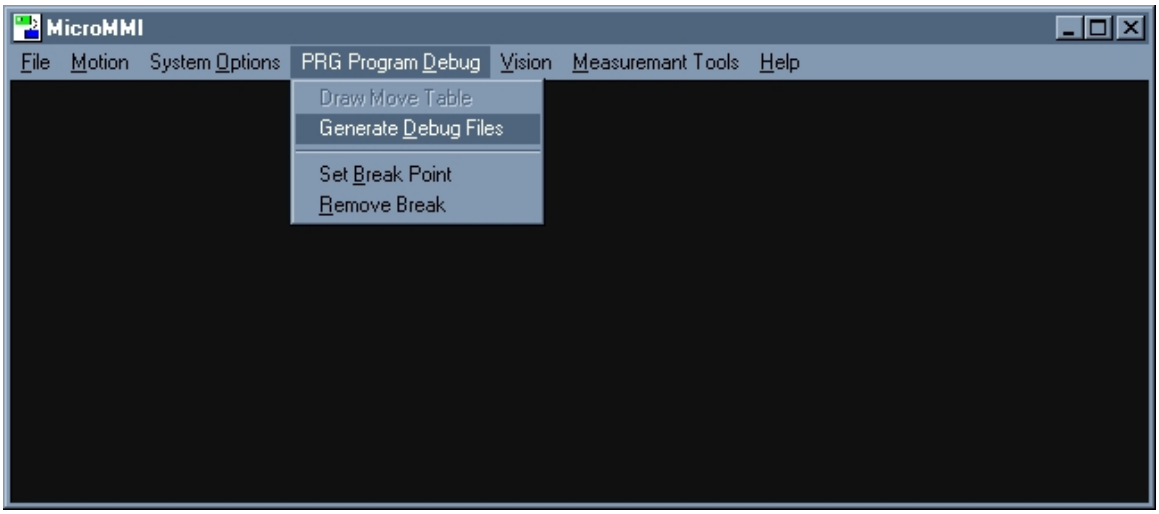
The calibration constants for both the X and the Y-axis are needed for the calibration dialog box, although they are usually the same value. Measure a known object in both vertical and horizontal directions to obtain the calibration constants for both axes.



At this time, Potomac is not using the Direction Sense, however, it will be used in a future application to further enhance the usability of the MicroMMI application.

12 Debug PRG Programs

There are several functions available for the operator during maintenance mode for debugging a PRG program before running it on the actual parts or hardware.



12.1 Generate Debug Files

When writing and debugging PRG programs it is often helpful to know exactly which PRG commands are being executed and more specifically which commands are being sent to the UNIDEX 500 controller card for execution. When the <Generate Debug Files> menu item is selected, three files are generated at the same time a PRG program is running. These files will be written to the same directory that the PRG program resides. Sometimes the files will be written to the directory that contains the MicroMMI application (D:\Potomac).

The first file that is generated is called U500Out.txt, which contains all PRG commands that are sent to the UNIDEX 500 motion controller for execution as opposed to being executed by the MicroMMI application itself. The next files is called PRGOut.txt and contains all lines of the PRG program that are translated and executed by the MicroMMI application in the exact order in which they are translated. That way, if a LOOP 3 PRG command is being executed, there will be three references to the PRG code contained within the LOOP command.

Below is an example of the prgout.txt file output:

Output of internal program execution during translation for> C:\u500\Projects\Hutchinson\Hutch1.prg
On> 12/17/99 5:21:50 PM
Passthrough Mode = True; Passing all MMI expressions directly to U500.
Pass Math Expressions = True; Passing all Math expressions directly to U500.
Use U500 Variables = True; Using the Variable values(V0-255) from U500.

V1=50

V2=0.16
V3=0.6879
V4=62.859
V6=10
V7=100
Going to subroutine: :TRAIN
so po x y
WA ON
v41=$xrp
v51=$yrp
Returning from subroutine: :TRAIN
Calling PPI function: #OVERLAY TYPE 2
PR IN ME UN UN/SE
TR LI
EN X Y
HO X Y
WA ON
SO PO X Y
SO HO
SCF X1 Y1
ROT X,Y,0
WA ON
G90
:START
OU 3,1
OU 5,1
OU 4,1
OU 0,1
OU 1,1
Calling PPI function: #STATUS "PROCESSING FIRST PLATE"

V80 = 486.530
V81 = 107.2
V85 = 549.373
V86 = 107.07
Going to subroutine: :PLATE
:PLATE
Going to subroutine: :TARGET
OU 2,0
Returning from subroutine: :TARGET
Executing LOOP start
Executing LOOP number 1
Executing LOOP start
Executing LOOP number 1
Going to subroutine: :DEBUS
Returning from subroutine: :DEBUS
Executing LOOP number 2
Going to subroutine: :DEBUS

Returning from subroutine: :DEBUS
Executing LOOP number 3
Going to subroutine: :DEBUS
Returning from subroutine: :DEBUS
Executing LOOP number 4
Going to subroutine: :DEBUS
Returning from subroutine: :DEBUS
…..
…..
…..
Executing LOOP number 10
Going to subroutine: :DEBUS
Returning from subroutine: :DEBUS
Returning from subroutine: :PLATE

It is very easy to see exactly where the execution flow is going. This file is helpful for control flow errors in the PRG program.

Below is the U500out.txt file output which corresponds to the above prgout.txt file output (you can tell by the time as all files are created at the same time and overwritten at the same time):

Output of actual statments sent to U500 during translation for> C:\u500\Projects\Hutchinson\Hutch1.prg
On> 12/17/99 5:21:50 PM
Passthrough Mode = True; Passing all MMI expressions directly to U500.
Pass Math Expressions = True; Passing all Math expressions directly to U500.
Use U500 Variables = True; Using the Variable values (V0-255) from U500.

V1=50
V2=0.16
V3=0.6879
V4=62.859
V6=10
V7=100
so po x y
WA ON
v41=$xrp
v51=$yrp
PR IN ME UN UN/SE
TR LI
EN X Y
HO X Y
WA ON
SO PO X Y
SO HO
SCF X1 Y1
ROT X,Y,0

WA ON
G90
OU 3,1
OU 5,1
OU 4,1
OU 0,1
OU 1,1
V80 = 486.530
V81 = 107.2
V85 = 549.373
V86 = 107.07
OU 2,0

These are the PRG commands that are actually sent to the UNIDEX 500 controller card for execution. All of the control flow commands that are interpreted strictly by the MicroMMI application are not written to this file. Both of the above files can also indicate where the program execution is stopping due to some error, either syntactical, logical, or mechanical.

The third file, IdOut.txt is not used at this time.

12.2 Set Break Point

The Set Break Point menu selection allows the operator, when in maintenance mode, to set a line number in the PRG program on which to stop execution. The operator must put the mouse on the line on which to stop and then click the mouse button. After the line has been selected, the menu selection can be made and that will set the line as a breakpoint line. Every time the program is executed, the execution will stop on this line. The only way to get rid of the break point line is to select the menu item <Clear Break Point> or select a different line as the break point line.

NOTE: Sometimes, if there are several comments around a line with a breakpoint set or several other non-executable lines, the program will stop on the line after the breakpoint.

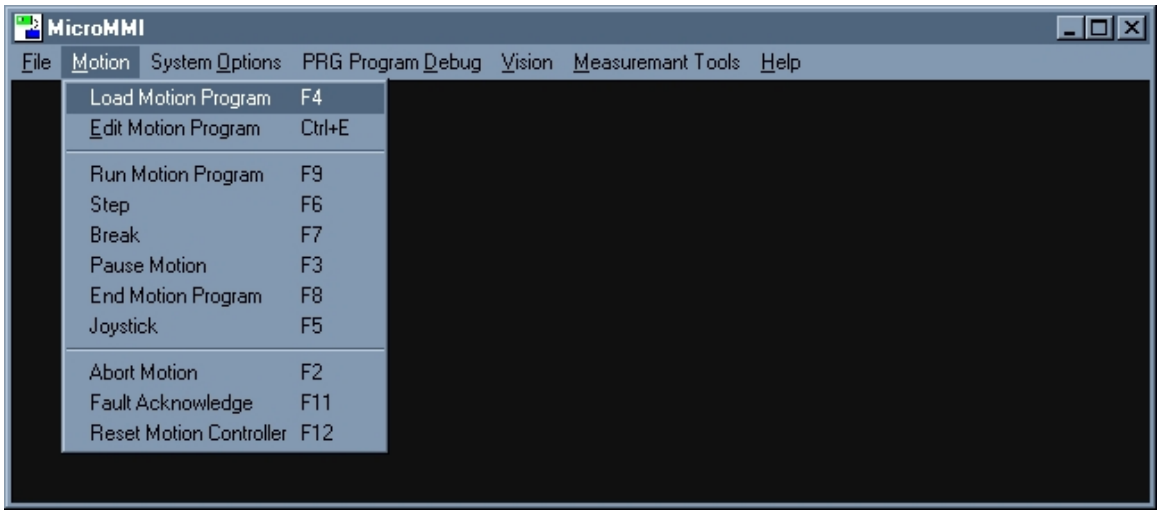
NOTE: At this time, there is no indication in the user interface, of a line that has a breakpoint associated with it. When in doubt of a breakpoint location, select <Debug> <Remove Breakpoint> to make sure there are no breakpoints set.

12.3 Remove Break Point

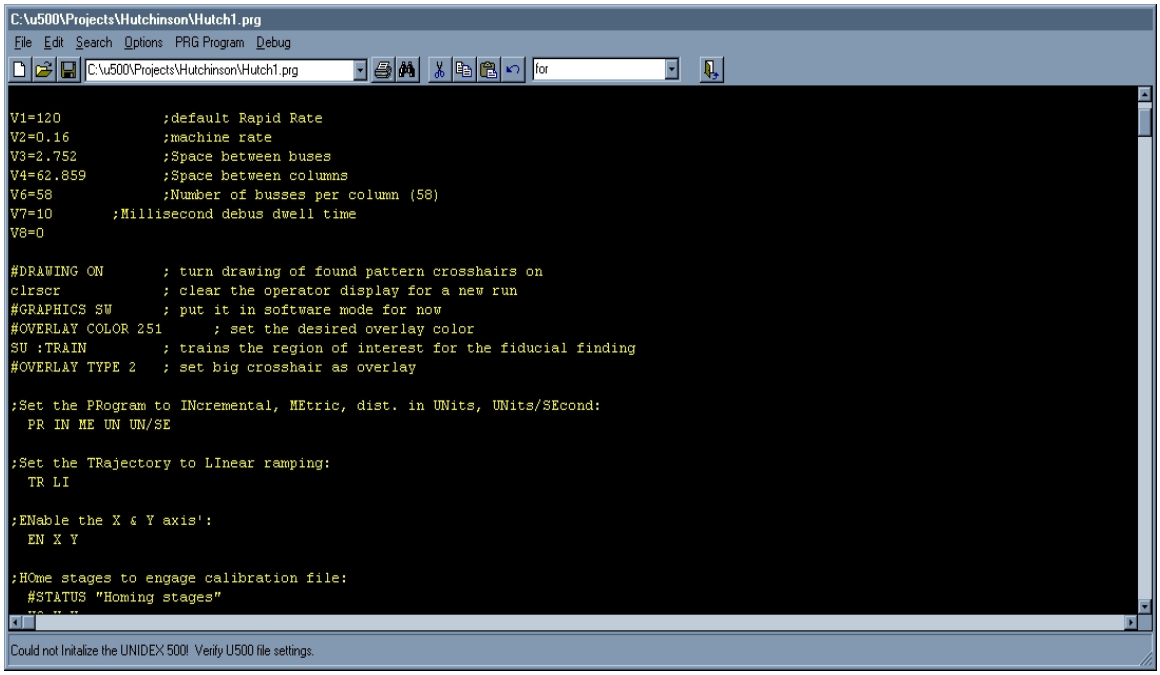
This menu selection clears the selected break point line. It does not matter where the selected line resides because there may be only one breakpoint set at a single time.

13 Edit Motion Program

The operator may invoke the PRG program editor, when in maintenance mode, by selecting the <Motion> <Edit Motion Program> menu selection. Those operators who have extensive knowledge of the motion control, programming process, usually bring up the PRG file editor. Using the editor interface has advantages over the MicroMMI interface when debugging a motion control program.



If the only requirement is to run a motion control program that performs the same operations over and over again, the best way to run MicroMMI would be to maintain the operator mode. That way, the security controls inherent in the software, will prevent the operator from accidentally modifying files or variables that could cause irreparable harm to the entire system.

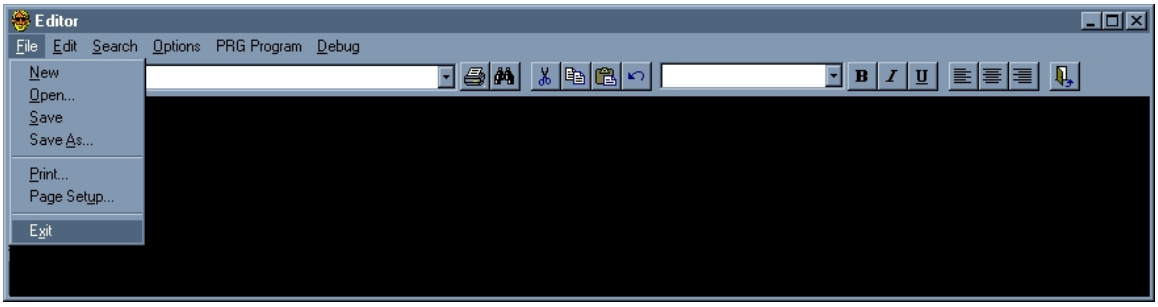


The MicroMMI application IDE provides the ability to open the editor while the program is running. This ability has been included to allow viewing of a larger amount of the PRG program while the program is running. It also allows the execution of the PRG program while in the editor so that more intensive PRG program debugging is available.

The editor included with MicroMMI is a typical editor with typical editing functions. There are several additions the typical editor functions, which include the ability to run and debug a PRG program directly from the editor.

13.1 Editor – File Menu

Either a new or an existing PRG program may be loaded into the editor. Be sure to validate all programs with the same name that reside in different directories. Programs may be Opened, Saved, Printed, and Closed.



The <File> <Open> menu selection opens the typical dialog box for the operator to select the PRG program in which to load. This load program dialog box does not save the value of the PRG program loaded for more than the existing MicroMMI execution. Only the <File> <Project> menu selection will bring up a window allowing the operator to save a PRG file name to the operating system's Registry, thereby always opening this PRG file when MicroMMI begins execution.

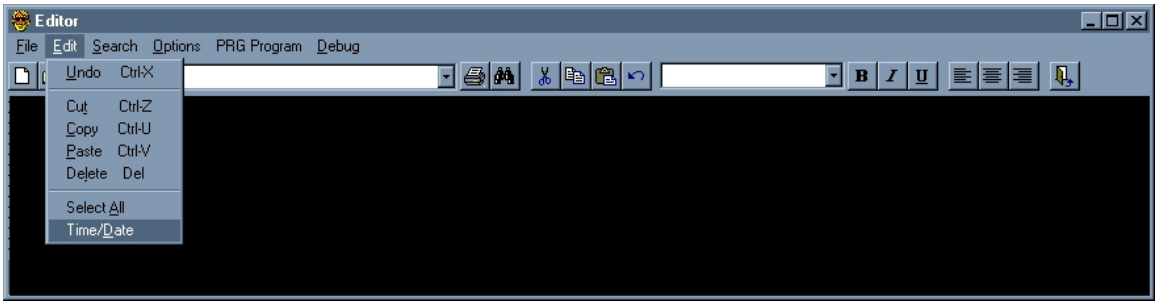
The rest of the menu selections on the <File> menu operate as would be expected in any program that has the ability to load and edit text files.

NOTE: The <Save> menu selection does not append the .PRG extension to the files if the operator does not explicitly type in that extension. If it is desired that the extension for a newly saved file should be .PRG, then the operator must type in the extension.

13.2 Editor – Edit Menu

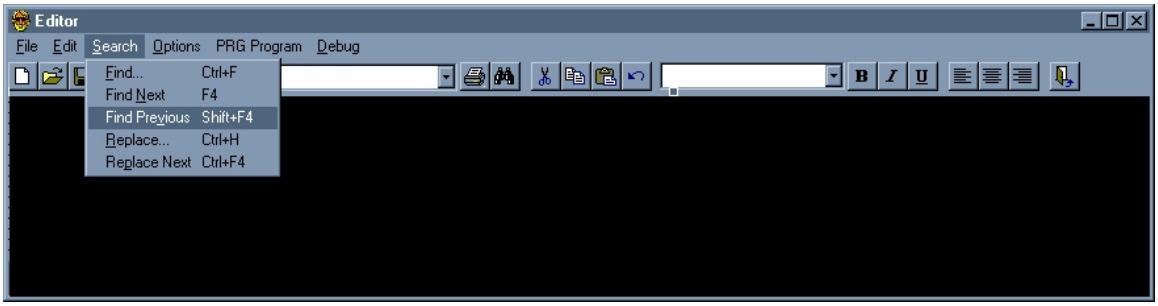
The general edit selections such as select, cut, paste, and copy are all available when MicroMMI is in maintenance mode.

NOTE: After having made changes to the PRG using the editing commands, the file is always saved automatically when the editor window is closed.

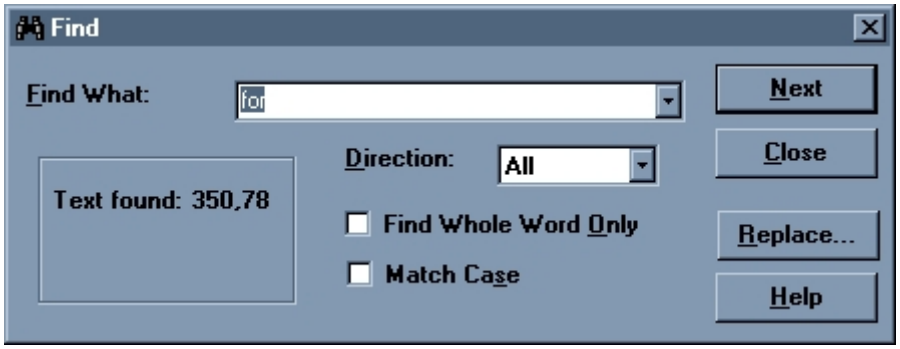


13.3 Search Menu Selection

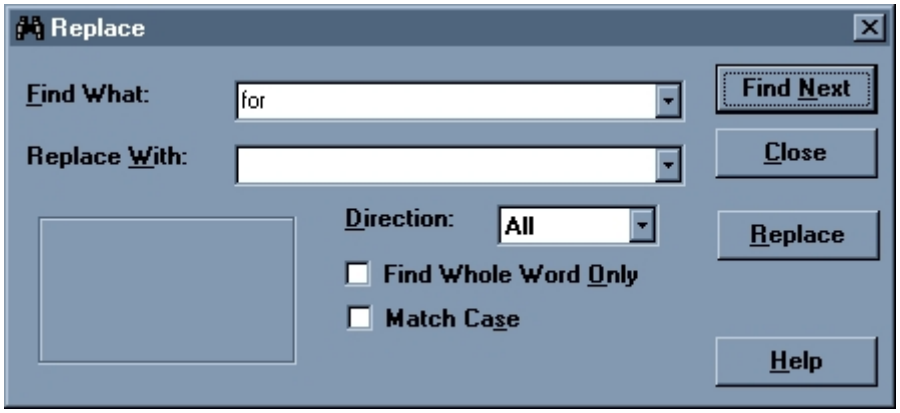
The <Search> menu selection brings up the screens for the search and replace functionality. The search and replace functions are similar to those always maintained by editors.



The find window is a non-modal window allowing the operator to search for multiple instances of the search string. After a search has been performed, the search string is also maintained in the toolbar of the editor for ultra-fast searches. The search edit box in the toolbar allows the operator to enter a new string to search. The search operation will use the existing search parameters of 'Find Whole Word Only', 'Match Case', and the direction.

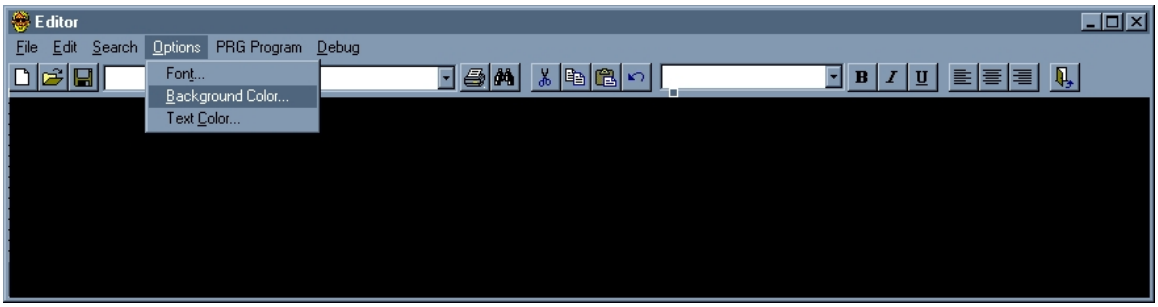


The replace operation can be used with the same parameters as the search. However, there is not ability to put a replace phrase on the main editor interface as there is with the search function.



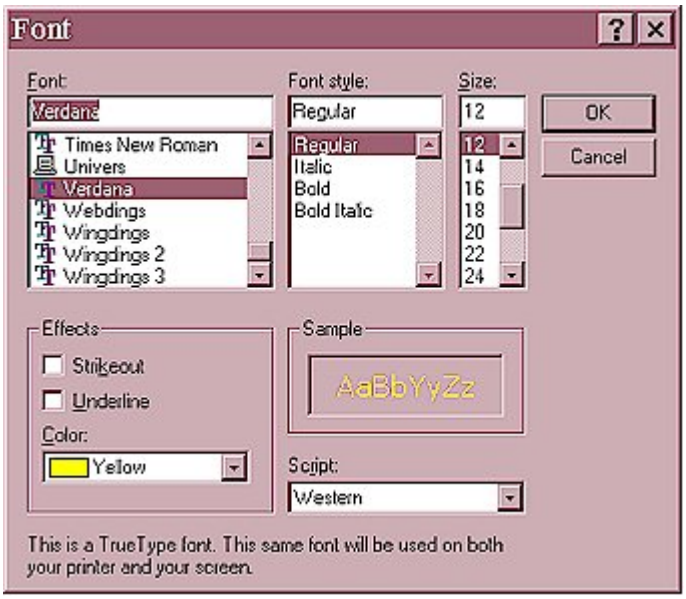
13.4 Options Menu Selection

The <Options> menu selection allows the operator to modify some of the editor's interface parameters such as text font, text color, background color, and the type of format in which the editor saves the file text.

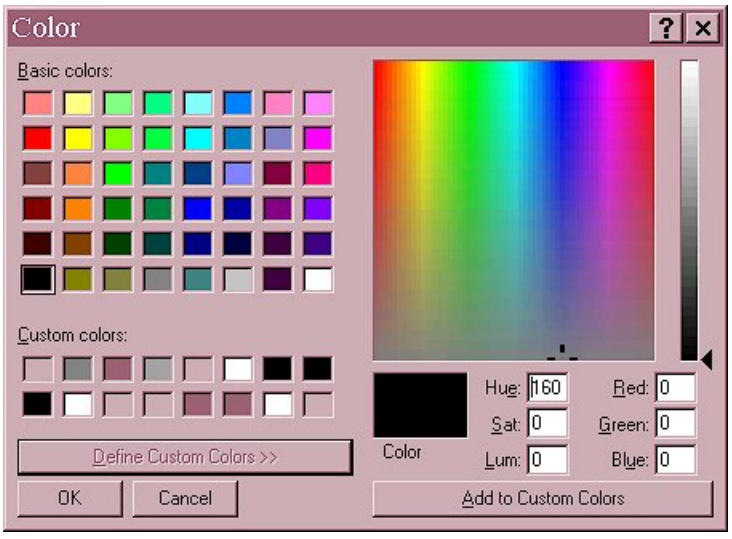


After the operator selects the <Options> <Font> menu item, the <Options> <Selection Font>, or the <Options> <Background Color> menu selection, the typical font dialog box will appear. The operator may then choose the font characteristics for the corresponding interface attribute.

NOTE: If the font size or any other characteristic that is selected causes the program lines to wrap around to the next line, the syntax of the program will be incorrect. PRG programming syntax does not allow word wrap so make sure that all lines begin and end on the same editor line or there will be errors during the Syntax Check operation.



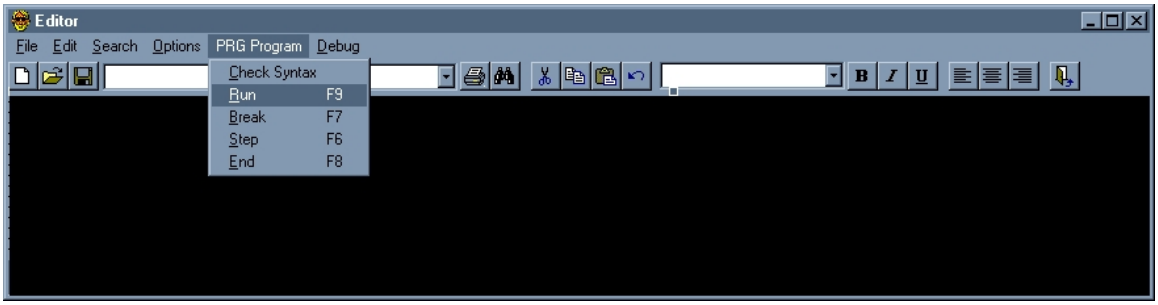
The default text font color is yellow. The default background color is black. The font point size has been set to 12 and the font name is Verdana. Verdana was chosen because of its ability to display properly the minus signs that often appear before numbers in a PRG motion control program.



The typical color dialog box is displayed when the operator chooses to change the color of either the text or the background. The operator may also specify custom colors using the Define Custom Colors button on the color dialog box window.

13.5 PRG Program Menu Selection

The <PRG Program> menu selection provides all of the execution capabilities that the <Motion> menu in MicroMMI provides and one more. The additional selection is the <Check Syntax> menu item where the operator may choose to verify the correctness of any changes made to an existing program or of a newly loaded program.



The <Run>, <Break>, <Step>, and <End> menu items maintain the same functionality as the execution buttons provided on the PRG Program Display area of the main interface window of MicroMMI.

13.5.1 Running PRG Programs

The loaded PRG program can be executed by pressing the <Run> menu item or by pressing F9. The loaded program will execute all valid MicroMMI instructions. Valid MicroMMI instructions are specified in Appendix A. The <Run> menu selection is available in all operating modes.

MicroMMI provides extended debugging functionality. Typical debuggers provide the following functions:

a) **Run** - runs the program from the first line of executable code until a break condition is met or until the end of the program is found

b) **Step** - steps through the program by executing one line of code per each selection of the step command (Step button or Step menu item)

c) **Break** - stops the execution of the program when it is running at whichever line of code the execution was on when the Break command was issued - allows the program to continue executing from the place where it stopped by the selection of the Run command (or continue command depending on the IDE interface)

d) **End** - prematurely stops the execution of the program and resets the program, thereby not allowing the user to continue execution like the Break command

This means that the IDE will be in one of the following states: EDITING, RUNNING, BREAK, or IDLE. When the MicroMMI application is first started the application is in the IDLE state. After the Run button is selected the program is in the RUNNING state. When the Break button is selected the application is in the BREAK state. After the PRG program execution is finished either by reaching the EXit command in the PRG program or after the user selects the End button, the application is in the IDLE state. Any time the user opens the editor and makes a change to the PRG source code, the MicroMMI application is in the EDITING state. The application will remain in that EDITING state until the PRG program is saved, when the application will revert back to the state that the application was in before it was edited.

13.5.2 Breaking a PRG Program

A PRG program can be stopped at any time during the execution of the program. After the program execution has been paused, it can be resumed by pressing the <Run> menu item. The <Break> menu selection is available only in maintenance mode.

NOTE: The laser should not be enabled in a program that is being debugged because the laser may stay too long in a particular location.

13.5.3 Stepping a PRG Program

A PRG program can be stepped in order to debug that program. For each <Step> menu item selection, a single line of PRG code will be executed. The <Run> menu item can be selected at any time during a stepping operation. If the <Run> item is selected, it will operate similar to a 'continue' operation in that the normal execution of the program will continue until the end of the program is reached or the operator selects the <Break> or <End> menu items. The Step button is available only in maintenance mode.

NOTE: If the <Step> command becomes unavailable during program execution, such as when it reaches a LOOP command, SUB command, etc., the <Break> button must be selected in order to continue the stepping operation.

NOTE: The laser should not be enabled in a program that is being debugged because the laser may stay too long in a particular location.

13.5.4 Ending a PRG Program

The program display will scroll along with the program's execution so that the operator may view the exact lines being executed. The <End> menu item can be selected at any time during the execution to completely stop the program. When the <End> menu item is pressed, the program may be restarted from the beginning by selecting the <Run> menu item. The <End> selection is available in all operating modes.

13.6 Debugging a PRG Program

For the experienced operator, there are special functions that are available in maintenance mode.

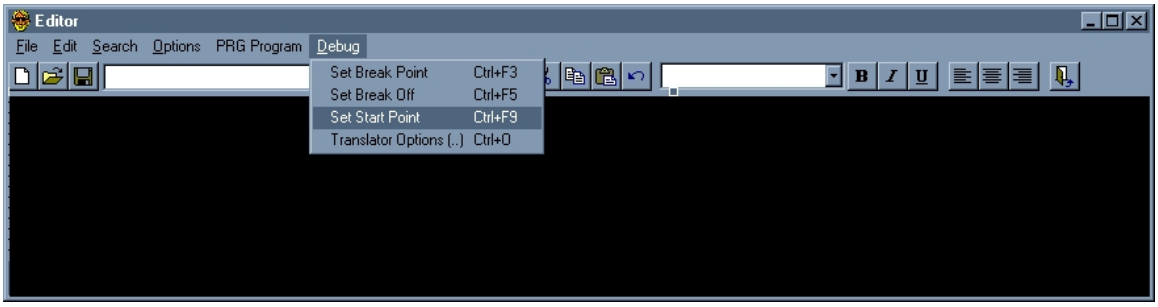
13.6.1 Setting a Break Point

A break point may be set on any valid PRG command line. When execution reaches that line, the execution stops and waits for the operator to select the <Run> menu item. If the operator has set a watch on a particular variable and the value is incorrect, the operator also has the option of ending the program to continue the edit of the program instructions.

NOTE: Sometimes the execution may stop on the after the one that was set for the breakpoint. This may be avoided by setting breakpoints on lines where there are no blank or commented lines around the desired breakpoint line.

The operator may set a break point before a set of instructions are reached in order to slow down the execution by stepping through those instructions that follow the break

point. In this way, the operator can closely watch the variable values and the control path through the instructions to verify their validity and correctness.



13.6.2 Removing a Break Point

After a section of instructions or command has been evaluated as correct, the operator would remove the break point to see if the program still runs correctly at run time speed. It may also be desirable to set another break point after the first has been reached, therefor executing the program in chunks.

13.6.3 Setting the Program Start Line

At times it is just easier to select the <End> menu item, especially if the program is in some king of infinite loop. However, the operator would not want to have to completely start execution from the beginning. It would be better to begin in the area where the program had just been stopped. For this there is an option of setting the start position. After the operator selects the starting line by clicking the mouse cursor on that line and then selecting the <Set Start Point> menu item, the program can continue and so can the debugging process.

NOTE: Any time the starting point is specified as something other than the correct sequence of execution there is the possibility that the program may no longer function as expected. For example, if the starting line were set after where the axes are enabled, then no move operations (or any other operations) that operate on the axes may be executed. An error message will be generated by the U500 (DR500) if there is a problem with the command.

Perhaps it is useful to change a variable value before continuing on with debugging. This would have to be performed while the program was not running. Then the operator can set the new start point and continue debugging after 'planting' the correct value for a variable.

13.6.4 Setting Translator Options

The translator options are for Potomac personnel who have extensive familiarity with the translator itself and Aerotech motion control programming. This option is not available to Potomac customers at this time.

14 Appendix A – PRG Programming Syntax

For a description of the specific syntax concerning the rules about programming the Aerotech UNIDEX 500 motion controllers, see the Aerotech documentation in the U500 directory of the Aerotech installation directory. The following information concerns what MMI commands have been implemented in this version of MicroMMI and what Potomac specific commands have been implemented.

14.1 IDENTIFIERS

Identifiers are strings of the LETTER_V,and a V_DIGIT which is a number from 0 through 255. An assignment statement always has an identifier as its left hand side (LHS). The right hand side (RHS) may be comprised of any expression or series of expressions. The input bit stream may be the RHS as well as the current axis position (feedback, relative, or absolute).

V*n* where *n* is an integer between 0 and 255

```
Identifier      →    LETTER _V  + V_DIGIT
```

Identifier Examples: V0, V12, V34, V250, etc.

Assignment Statement Examples: V12 = (V1 + 3) / ABS(V4), V12 = 43.2, V12 = $IN3, V12 = $2FP, etc.

14.2 OPERATORS

14.2.1 Math Operators:

These operators operate on two operands, resulting in a single operand result. Any group or combination of these operations can occur and the proper calculation of precedence will be performed. These operations can occur in any assignment statement. An assignment statement is an assignment of a value to an identifier, in this case, the Aerotech variables (V0-255). NOTE: Mathematical operations are not supported as a component, or sub-expression of other statement types unless the syntax checking has been turned off and the Translator is in Pass-Through Mode.

- `( )  =  Grouping`
- `*  =  Multiplication`
- `/  =  Division`
- `+  =  Addition`
- `-  =  Subtraction`
- `=  =  assignment`

14.2.2 Unary Operators:

Unary operators are operators that operate on one operand, resulting in a single operand result. Unary operations may exist alone, in a single assignment statement, or may be embedded within a group of mathematical operations as defined above. NOTE: Unary

operations are not supported as a component, or sub-expression of other statement types unless the syntax checking has been turned off and the Translator is in Pass-Through Mode.

- NOT(val) = inverse value of “val”
- ABS(val) = absolute value of “val”
- SQR(val) = square root of “val”
- DEG(*radians*) Convert radians to degrees (NOT IMPLEMENTED YET)
- RAD(*degrees*) Convert degrees to radians (NOT IMPLEMENTED YET)

(all angles in *radians*)

- TAN(*angle*) = tangent
- ATN(*angle*) = arctangent
- SIN(*angle*) = sine
- ASIN(*angle*) = arcsine
- COS(*angle*) = cosine
- ACOS(*angle*) = arccosine

The following unary expressions can not be used within Aerotech’s MMI because they are additions to the language specifically for this program. Therefore, in order to use te following operators, the program can not be in Pass-Through Mode.

- RAN(val) = returns a random number using “val” as the key
- EXP(val) = “val” raised to the power of 2
- LOG(val) = natural log of “val
- SEC(angle) = secant
- ASEC(angle) = arcsecant

14.2.3 Relational, Comparison, or Boolean Operators:

Relational operators operate on two operands, resulting in a single temporary operand result. Relational operations are usually part of a control structure, such as an iteration statement, or a conditional branching statement. The operand result of the conditional expression is used to determine whether a particular set of statements should be executed or not executed. Relational expressions are not to be used within a mathematical assignment statement, but only as the test condition for a conditional branch or other control statement.

- = = “EQ” equal to
- < = “LT” less than
- > = “GT” greater than
- <> = “NE” not equal to
- >= = “GTEQ” greater than or equal to
- <= = “LTEQ” less than or equal to

14.2.4 Control Operators:

Control operators are comprised of branching, jumping, looping, and iteration statements. These statements are all valid control structures in the Aerotech PRG programming language. See the Aerotech documentation for further information.

```
<selection_statement>
     →   "IF" <relational_expression> <label>
     →   "IF" ( <relational_expression> ) <label>
     →   "IF" ( <relational_expression> ) "THEN"
              <statement_list>
         "ENDIF"
     →   "IF" ( <relational_expression> ) "THEN"
              <statement_list>
         "ELSE"
              <statement_list>
         "ENDIF"

<iteration_statement>
     →   "WHILE"   <relational_expression>
                   <statement_list>
         "ENDWHILE"
     →   "LOOP" Integer
              <statement_list>
         "NEXT"
     →   "LOOP" Identifier
              <statement_list>
         "NEXT"

<jump_statement>
     →    "GOTO" <label>
     →    "SUBROUTINE" <label>
```

14.3 Aerotech MMI Commands

G-Code	MMI Name	Abbreviation	Interpreted	MicroMMI Supported
---	AutoTuning	AT	No	YES
G0	INDEX	IN	No	YES†
G1	LINEAR	LI	No	YES†
G2	CW_CIRCLE	CW	No	YES
G3	CCW_CIRCLE	CC	No	YES
G4	DWELL	DW	Yes	YES
G8	VELOCITY ON	VE ON	No	YES
G9	VELOCITY OFF	VE OFF	No	YES
G23/G24	ROUNDING ON/OFF	ROUNDING ON/OFF	No	YES
G70	PROGRAM ENGLISH	PR EN	No	YES

G71	PROGRAM METRIC	PR ME	No	YES
G90	PROGRAM ABSOLUTE	PR AB	No	YES
G91	PROGRAM INCREMENTAL	PR IN	No	YES
G92	SOFTWARE HOME	SO HO	No	YES
M0	pause	---	Yes	YES
M2	EXIT	EX	Yes	YES
M47	AGAIN	AG	Yes	YES
---	BOARD	BO	No	YES
---	ABORT	AB	Yes	YES
---	Fault Acknowledge	FA	Yes	YES
---	CLEAR SCREEN	CLRSCR	Yes	YES
---	DISABLE	DI	No	YES
---	ENABLE	EN	No	YES
---	GOTO :label_marker	GO :(label)	Yes	YES
---	HOME	HO	No	YES
---	IF	IF	Yes	YES
---	LOOP / NEXT	LO / NEXT	Yes	YES
---	MAP	MA	No	YES
---	MESSAGE	ME	Yes	YES†
---	INPUT	IN / INP	Yes	YES†
---	OUTPUT	OU	Yes	YES
---	SCALE FACTOR	SCF	No	YES
---	ROTATION	ROT	No	YES
---	Software Key	SK / SKEY	Yes	YES†
---	SLEW	SL	No	YES
---	SPLINE	SP	No	YES
---	SOFTWARE POSITION	SO PO	No	YES
---	SUBROUTINE / RETURN	SU / RE	Yes	YES
---	VARIABLES	VAR	Yes	NO
---	WAIT ON/OFF/ALL	WA ON/OFF/AL	Yes	YES
---	UMFO	UMFO	No	YES
---	Target tracking	TE | TD | TP	No	YES
---	BRAKE	BR	No	YES
---	FiLter time constant	FL	No	YES
---	FREERUN	FR	No	YES
---	GAIN	GA	No	YES
---	GEAR	GE	No	YES
---	LVDT	LVDT	No	YES
---	Motor Communication	MC	No	YES
---	Motor SETup	MSET	No	YES
---	PLANE	PL	No	YES
---	PSOP	PSOP	No	YES

---	PSOD	PSOD	No	YES
---	PSOF	PSOF	No	YES
---	PLC	PLC	No	YES
---	RAMP	RA	No	YES
---	QUEUE	QU	No	YES
---	SEGMENT	SE	No	YES
---	WHILE / ENDWHILE	WH / ENDW	Yes	YES
---	TRAJECTORY	TR	No	YES
G40	cutter comp. off	---	No	YES
G41	cutter comp. on, *left*	---	No	YES
G42	cutter comp on, *right*	---	No	YES
G43	define cutter radius	---	No	YES
G44	define compensated axes	---	No	YES
---	ACCELERATION	AC	No	YES
---	CYCLE ON/OFF	CY ON/OFF	No	YES
---	HALT	HA	No	YES
G0	INDEX	IN	No	YES†
G1	LINEAR	LI	No	YES†
G2	CW_CIRCLE	CW	No	YES
G3	CCW_CIRCLE	CC	No	YES
G4	DWELL	DW	Yes	YES
G8	VELOCITY ON	VE ON	No	YES
G9	VELOCITY OFF	VE OFF	No	YES
G23/G24	ROUNDING ON/OFF	ROUNDING ON/OFF	No	YES
G70	PROGRAM ENGLISH	PR EN	No	YES
G71	PROGRAM METRIC	PR ME	No	YES
G90	PROGRAM ABSOLUTE	PR AB	No	YES
G91	PROGRAM INCREMENTAL	PR IN	No	YES
G92	SOFTWARE HOME	SO HO	No	YES
M0	pause	---	Yes	YES
M2	EXIT	EX	Yes	YES
M47	AGAIN	AG	Yes	YES
	ABORT	AB	Yes	YES
	Fault Acknowledge	FA	Yes	YES
---	CLEAR SCREEN	CLRSCR	Yes	YES
---	DISABLE	DI	No	YES
---	ENABLE	EN	No	YES
---	GOTO :label_marker	GO :(label)	Yes	YES
---	HOME	HO	No	YES
---	IF	IF	Yes	YES
---	LOOP / NEXT	LO / NEXT	Yes	YES
---	MAP	MA	No	YES

---	MESSAGE	ME	Yes	YES†
---	INPUT	IN / INP	Yes	YES†
---	OUTPUT	OU	Yes	YES
---	SCALE FACTOR	SCF	No	YES
---	ROTATION	ROT	No	YES
---	Software Key	SK / SKEY	Yes	YES†
---	SLEW	SL	No	YES
---	SPLINE	SP	No	YES
---	SOFTWARE POSITION	SO PO	No	YES
---	SUBROUTINE / RETURN	SU / RE	Yes	YES
---	VARIABLES	VAR	Yes	NO
---	WAIT ON/OFF/ALL	WA ON/OFF/AL	Yes	YES
---	WHILE / ENDWHILE	WH / ENDW	Yes	YES
G40	cutter comp. off	---	No	YES
G41	cutter comp. on, *left*	---	No	YES
G42	cutter comp on, *right*	---	No	YES
G43	define cutter radius	---	No	YES
G44	define compensated axes	---	No	YES
	ACCELERATION	AC	No	YES
	CYCLE ON/OFF	CY ON/OFF	No	YES
	HALT	HA	No	YES

Notes:
† Not all arguments/syntax supported at this time

14.4 Potomac Proprietary Command Format

The Potomac Proprietary (PPI) commands are single word commands that are prefixed by a hash (pound) sign.

Command	**parameter**	**parameter**	**parameter**	**parameter**
#BEEP	Integer number of beeps			
#IMAGE REDRAW	none			
#IMAGE LOAD	Filename			
#IMAGE SAVE	Filename			
#IMAGE TRAIN	Roi_name	quality		
#GRAPHICS	HW / SW			
#DRAWING	ON / OFF			
#OVERLAY TYPE	Integer			
#OVERLAY COLOR	Integer			
#ERODE	none			

#DILATE	none											
#SHARPEN	none											
#BRIGHTEN	none											
#THRESH AUTO	sensitivity	level		smooths				Sub X	Sub Y			
#THRESH BILEVEL	Threshold value											
#FIND EDGE	Roi_nam	Direction		polarity		strength		=	X	Y		
#FIND BLOB	Roi_nam	Min width	Max width	Min height		Max height		color	= X Y			
#FIND PATTERN	Roi_name	Quality				=		CentX	CentY			
#ROI TRAIN	Roi_name	Quality										
#ROI SHOW	Roi_name											
#ROI HIDE	Roi_name											
#ROI ADD	Roi_name	XStart	YStart	XLength		YLength	Visible					
#ROI DELETE	Roi_name											
#FIRE	None	None		None		none						
SKEY	PPI	integer key number		string command		string text for key						

14.4.1 Machine Vision Proprietary Commands

The machine vision subsystem allows the user to perform rudimentary image processing and measurement of features on live or still images. The output from this processing may then be used to inspect the progress of part processing or make a decision based on some feature visible via the video subsystem. Commands were added to the standard G-Code syntax, which allows the user to use the same PRG program to also apply the machine vision aspect of the application software.

There are many system variables that will affect the operation and successful application of the machine vision. These include, but are not limited to, system lighting, geometrical aspects of the part under inspection, image background noise, near field obstructions and density of features, focus and magnification of optics, and computer system loading.

MicroMMI applies standard image processing algorithms to 8 bit, 256 grayscale images that are captured via the video subsystem. Still images that have not been captured by the video subsystem may be loaded and examined as long as they are in an 8 bit, 256 grayscale, and bitmapped format. The following will explain the different functions that may be applied to an image captured by the live video subsystem.

There are two kinds of images in MicroMMI, master images and rectangular regions of interest (ROIs) that exist inside master images. The master image corresponds to the entire video display area in the main user interface of MicroMMI. Both kinds of images are described similarly and so appear nearly identical to the MicroMMI image processing functions. For master images, the starting location of the image (x,y) is always 0,0 and is referred to as the image origin. For ROIs, the starting x,y location, or origin, is relative to

the origin of the master image. In both master and ROI images, dx,dy is the size of the image in the X (horizontal or length) and Y (vertical or height) directions.

The master image has x,y = 0,0 and size dx,dy=640,480. The next two specify ROIs. In this example, neither ROI has an x,y = 0,0 although it is possible for the ROI to begin at the master image origin. The ROI dx,dy size must be the same or smaller than the master. ROIs may be placed inside or overlapping another ROI but they are always referenced relative to the master image and no information is retained to indicate they are nested or overlapping.
ROIs are used to define the region within an image in which a function will be applied. ROIs limit the size of the area that must be operated upon when applying a function. The smaller the size of the ROI, the less time a function will take to complete its operation. Therefor, ROIs should be made as small as possible for optimal execution speed.

ROIs may be added, deleted, hidden, and displayed according to the following PRG program syntax.

14.4.2 ROI ADD

For an ROI to be added to the current list of ROIs, the user must specify where the ROI is to be placed, a unique name for the ROI, and whether or not an outline of the ROI should be drawn to the live video display.

#ROI ADD NameOfROI XStart YStart XLength YLength Visible

Parameters

NameOfROI	-	**String indicating a unique name for the ROI**
Xstart	-	**Starting X location (upper, left corner) of ROI.**
Ystart	-	**Starting Y location (upper, left corner) of RROI.**
Xlength	-	**Horizontal size of the RROI.**
Ylength	-	**Vertical size of the RROI.**
Visible	-	**Boolean indicating whether to show the drawn ROI.**

Returned Values

There is no returned value for the PRG programmer to use however, the user will receive an error message if the name of the ROI conflicts with an ROI of the same name which has already been added to the list of ROIs.

14.4.3 ROI DELETE

Any defined ROI may be deleted. The only parameter is the name of the ROI to be deleted. The user will receive an error message if the ROI does not exist. The ROI will be removed from the live video display.

#ROI DELETE NameOfROI

Parameters

NameOfROI - String indicating a unique name for the ROI

Returned Values

There is no returned value for the PRG programmer to use however the user will receive an error message via a dialog box if the ROI does not exist.

14.4.4 ROI HIDE

Any previously defined ROI may be hidden. The only parameter is the name of the ROI to be hidden. The user will receive an error message if the ROI does not exist. The ROI will be hidden from the live video display.

#ROI HIDE NameOfROI

Parameters

NameOfROI - String indicating a unique name for the ROI

Returned Values

There is no returned value for the PRG programmer to use however the user will receive an error message via a dialog box if the ROI does not exist.

14.4.5 ROI SHOW

Any previously defined ROI may be displayed in the live video. The only parameter is the name of the ROI to be shown. The user will receive an error message if the ROI does not exist. The ROI will be drawn to the live video display.

#ROI SHOW NameOfROI

Parameters

NameOfROI - String indicating a unique name for the ROI

Returned Values

There is no returned value for the PRG programmer to use however the user will receive an error message via a dialog box if the ROI does not exist.

14.4.6 ROI TRAIN NameOfROI QualityValue

A Pattern Finding tool, is a machine vision tool that measures and reports the location of a previously trained pattern in an image. A pattern is a region of interest (in an image) that contains meaningful and unique information. A pattern can be as simple as an edge with few pixels, or a very complex pattern of thousands of pixels, such as human face. Typically, a pattern is trained once and searched for in other images. This function is designed to overcome real-world problems, such as rotated, scaled or occluded pattern, missing part of pattern, non-linear reflectance, contrast reversal, oxidization, deposits, and contour type images. Changes occur in a part, caused by use (wear) or by manufacturing process. Some common examples are changes in reflectance, integrity (missing part of the pattern), scale, and mechanical positioning. The light source used to illuminate the part could change with age, or the intensity of the light source may vary with respect to the position of the part. All of these would make the search pattern differ from the trained model. Some types of imaging devices, such as electron microscopes, produce images with a high noise level, and exhibit a non-linear reflectance behavior. However, with high-level noise, the noise pixels could be picked up as real pattern pixels. Some pre-processing is recommended, such as eroding (low-pass filtering) or median filtering, before training and searching for the pattern. Parts or objects with smooth or polished surfaces reflect light. Very slight changes in the angle or intensity of the light source results in very large changes in the gray-scale intensity levels. Under inspection conditions, light is often reflected unevenly (non-linearly). Dealing with non-linear reflectance has always been a problem in machine vision.

Any geometrical pattern may be trained and then searched for later in the application. The pattern must be trained before it can be found. At this time, there is no ability to save the trained pattern over program instances therefor the pattern must be trained at the beginning of each PRG program execution. Once trained, the pattern may be found in any number of frames of image data, either live or loaded into the display via the Image Load function.

Any previously defined ROI may be trained. The only parameter is the name of the ROI to be trained. The user will receive an error message if the ROI does not exist. The ROI will be hidden from the live video display.

#ROI TRAIN NameOfROI QualityValue

Parameters

NameOfROI - String indicating a unique name for the ROI

QualityValue - An integer value indicating the quality of the pattern that is being trained.

Returned Values

There is no returned value for the PRG programmer to use however the user will receive an error message via a dialog box if the ROI does not exist.

14.4.7 FIND EDGE

Any previously defined ROI may be used with the FIND EDGE function. FIND EDGE finds an extended edge in the ROI. If the edge is approximately horizontal, set Direction to 0 (zero). If the edge is approximately vertical, set Direction to 1 (one). Horizontal or vertical means the direction of the tangent to the edge. The ROI is then scanned in a direction normal (perpendicular) to the edge looking for the edge transitions.
Specify the Polarity of the transition as TRUE, then you are looking for an edge that is dark to light. If Polarity is FALSE, then you are looking for an edge that is light to dark. Edge transitions must be stronger than Strength to be considered.

The set of edge transitions that match the specified Polarity and that are stronger than Strength are used in a linear regression line fit, to compute the equation of the line that is parallel to and at the edge.

An edge with the characteristics associated with the parameters will be found in the specified ROI. Only one edge will be found in the ROI. If there are many edges defined in the ROI than the results of the edge detect function will be undetermined.

#FIND EDGE NameOfROI Direction Polarity Strength = CentX CentY

Parameters

NameOfROI - String indicating a unique name for the ROI
Direction - Horizontal = 0 or Vertical = 1 indicates the direction of the expected edge.
Polarity - Dark to Light = 0 or Light to Dark = 1 indicates the pixel intensity transition.
Strength - Value between 0 and 100 indicating the minimum strength of transition to qualify as an edge.

Returned Values

CentX - The X coordinate of the midpoint of the found line.
CentY - The Y coordinate of the midpoint of the found line.

The coordinate of the found edge is returned to the user. This coordinate will be calibrated according to the calibration factors that the user has previously specified in the MicroMMI program.

The user will receive an error message via a dialog box if the ROI does not exist.

14.4.8 FIND BLOB

Searches the region of interest and finds “blobs”, connected areas of pixels with the same “color”. Blob analysis is used to segment an image into ”islands” of pixels with uniform intensity, and then to measure characteristics of each island. Any previously defined ROI may be used with the FIND BLOB function. A blob with the characteristics associated with the parameters will be found in the specified ROI. Only one blob will be found in the ROI. If there are more than one blobs then only the first blob found with respect to the origin of the ROI will be found.

The Blob function will always apply an automatic threshold function for a low contrast image before searching for the blob. It is not necessary to first apply the threshold before calling this function.

The tuning application, MicroCaliper, will limit the difference between the minimum and maximums in both width and height to 50 pixels. This way, the blob function searches for blob of a very specific area. If it is desired to find any blob of any size, change the values of the minimum and maximum parameters so that the minimum size is quite small and the maximum size is quite large.

#FIND BLOB NameOfROI MinWidth MaxWidth MinHeight MaxHeight Color = CentX CentY

Parameters

NameOfROI - String indicating a unique name for the ROI
MinWidth - Blobs with width (X size) less than MinWidth will be ignored..
MaxWidth - Blobs with width (X size) greater than MaxWidth will be ignored.
MinHeight - Blobs with height (Y size) less than MinHeight will be ignored.
MaxHeight - Blobs with height (Y size) less than MaxHeight will be ignored.
Color - Value that indicates the expected grayscale value of the blob where 0 = white and 255 = black.

Returned Values

CentX - X location of the center of the blob.
CentY - Y location of the center of the blob.

The coordinates of the first found blob are returned to the user. These coordinates will be calibrated according to the calibration factors that the user has previously specified in the MicroMMI program.

The user will receive an error message via a dialog box if the ROI does not exist.

14.4.9 FIND PATTERN

The search will take place in the current display area. Only one pattern may be trained at one time, therefor only one pattern may be searched for at a single time. Only a single instance of the found pattern is returned at this time.

Any previously trained ROI may be found. The only parameter is the name of the ROI to be found. The user will receive an error message if the ROI does not exist. The user will also receive an error message if no pattern has been trained.

 # FIND PATTERN NameOfROI QualityValue = CENT_X CENT_Y

Parameters

NameOfROI - String indicating a unique name for the ROI
QualityValue - Integer between 0 and 100 indicating the required conformance to the trained pattern. A value of 10 would be very little conformance whereas a value of 98 would require an exact match (high conformance).

Returned Values

CentX - X location of the center of the pattern.
CentY - Y location of the center of the pattern.

The coordinates of the found pattern are returned to the user. These coordinates will be calibrated according to the calibration factors that the user has previously specified in the MicroMMI program.

There is no returned value for the PRG programmer to use however the user will receive an error message via a dialog box if the ROI does not exist.

14.4.10 THRESH AUTO

This is an automatic thresholding function which analyzes subregions of the master image whose size is defined by SubX and SubY. Each region is analyzed to determine whether Sensitivity percent of the pixels in the region have average brightness values that are separated by a range of 25 gray levels. If so, a threshold is determined from a statistical analysis of the region. This threshold may be biased high or low by the Level

parameter, where a level of 0 corresponds to a threshold at the lower likely threshold limit, and 100 corresponds to the upper likely limit.

In the event that no threshold can be determined for a subregion, the subregion is classified as low-contrast, and its threshold is determined based on the threshold determined by its neighbors. The resulting variable threshold function is smoothed Smooths times and applied across the entire image. Smoothing the threshold one or two times is almost always a good idea.

This function normally works best on two-intensity images. Its advantage is that the two intensities may vary from one picture to the next (Over Time), or from one image area to the next (over Space). Define subregion sizes (SubX, SubY) to be 1-2 times the size of the smallest feature which you wish the thresholding to find. For example, if you are thresholding 30x60 pixel characters, a 30x60 grid should be fine. If you're thresholding out 5x5 connector pin tips, try a 10x10 grid.

This function is useful for setting images to a considerably high contrast representation whereby an edge detect or a blob function can more easily be applied.

#THRESH AUTO Sensitivity Level Smooths SubX SubY

Parameters

NameOfROI - String indicating a unique name for the ROI
Sensitivity - Percent of the pixels in the region have average brightness values that are separated by a range of 25 gray levels
Level - Threshold may be biased high or low by the Level parameter, where a level of 0 corresponds to a threshold at the lower likely threshold limit, and 100 corresponds to the upper likely limit
Smooths - Resulting variable threshold function is smoothed Smooths times and applied across the entire image
SubX - Width of subregion
SubY - Height of subregion

Returned Values

There is no returned value for the PRG programmer to use.

14.4.11 THRESH BILEVEL

Thresholds image, converting it into a two level (bi-level) image. Pixel values in image below or equal to thresh are set to value below. Pixel values above thresh are set to pixel value above.

This function is useful for setting images to a considerably high contrast representation whereby an edge detect or a blob function can more easily be applied.

#THRESH BILEVEL Thresh

Parameters

Thresh - Value at which all pixels less than will be forced to 0 and all pixels greater than will be forced to 255.

Returned Values

There is no returned value for the PRG programmer to use.

14.4.12 IMAGE LOAD

Any bitmap file may be loaded into the current display and all available functions may be used on the newly loaded image. The image can be subsequently saved to either the same or to a different file name.

The image MUST be an 8 bit, 256 grayscale image for the image to be recognized by this program. If the image does not meet the necessary specifications for loading into MicroMMI, Microsoft's Photo Editor (and many other programs) may be used to convert the image into the correct format before loading.

#IMAGE LOAD Filename

Parameters

Filename - String indicating a file name to be loaded into the current display

Returned Values

There is no returned value for the PRG programmer to use.

14.4.13 IMAGE SAVE

The current display may be saved to a bitmap and several other file formats. These images may be subsequently reloaded into MicroMMI or used for any other purpose. The images are saved as 8 bit, grayscale images.

#IMAGE SAVE Filename

Parameters

Filename - String indicating a file name into which to save the current display

Returned Values

There is no returned value for the PRG programmer to use.

14.4.14 IMAGE REDRAW

The current display may be redrawn if for any reason the display has not fully updated or another window has caused part of the current display window to be cleared.

Redraw is also helpful when a message box has been displayed on top a still video image and the image has lost its color palette. The color palette is often rewritten by other Window-based applications running concurrently with the MicroMMI program. Because MicroMMI performs image processing on monochromatic images containing 8 bits, there is the limitation that the program must be run using the Windows display setting of 256 colors. When Windows is running in 256 color mode, palettes are used to draw to the monitor's display screen. When a palette that contains different colors than what is currently being used for the image processing, the previously grayscale image may turn to multicolor. When this happens, REDRAW can be used to recover the correct display palette.

#IMAGE REDRAW

Parameters

None

Returned Values

There is no returned value for the PRG programmer to use.

14.4.15 BRIGHTEN

This function will significantly brighten up an image that is very dark. If the image also has low contrast, some type of threshold with an ERODE may be helpful also.

#BRIGHTEN

Parameters

None

Returned Values

There is no returned value for the PRG programmer to use.

14.4.16 SHARPEN

This function will sharpen an image that has low contrast resulting in indistinguishable edges.

#SHARPEN

Parameters

None

Returned Values

14.4.17 DILATE

This function will enlarge the dark areas in an image. This function is useful when dark edges in an image are very difficult to find.

#DILATE

Parameters

None

Returned Values

14.4.18 ERODE

This function will enlarge all of the light areas in an image. Eroding an image is helpful when the threshold function has reduced the accuracy of placement of the features in an image.

#ERODE

Parameters

None

Returned Values

14.4.19 OVERLAY TYPE

This function will change the type of overlay graphic that is being displayed to the live video display area or it will allow the operator to draw a graphical overlay to that area.

#OVERLAY TYPE OverlayTypeNumber

Parameters

OverlayTypeNumber - Integer number corresponding to the type of overlay to draw

Returned Values

14.4.20 OVERLAY COLOR

This function will change the type of overlay graphic that is being displayed to the live video display area or it will allow the operator to draw a graphical overlay to that area.

#OVERLAY COLOR OverlayColorNumber

Parameters

OverlayColorNumber - Integer number corresponding to the color of which to draw the overlay graphics

Returned Values

14.4.21 DRAWING

This function will allow the operator to turn the drawing of the found lines, blobs, and patterns on or off depending on the function parameter.

#DRAWING On/Off

Parameters

On / Off - On indicates that all lines, blobs, and patterns found will be depicted with graphics / Off indicates that no graphics will depict the found items.

Returned Values

14.4.22 GRAPHICS

This function will allow the operator to specify whether the live video update and the overlay drawing will be performed with full hardware acceleration or by software emulation. There are times that full hardware acceleration may cause certain problems with the live video display. As we at Potomac are remedying all potential situations, this

command when used to set full hardware acceleration, should be used with care. If there are any problems noticed when using full hardware emulation, choose software instead and inform Potomac Photonics of the situation.

#GRAPHICS HW / SW

Parameters

SW - Software emulation
HW - Full hardware acceleration.

Returned Values

15 Appendix B – Formal Grammar Specification in BNF

15.1 Syntax Notation

```
→            = defined as
|            = OR
+            = AND (union)
[ ... ]      = is an operator which means `optionally'
{ ... }      = operator which means `one or more times'
```

Any time the symbol, →, is seen in an expression, it means that the left-hand-side (LHS) may be replaced with the right-hand-side (RHS). For example, see the following:

```
<expression> →    <term> <mult_operator> <term>
             →    <term> <add_operator> <term>
```

The expression on the second line (<term><add_operator><term>) can be used to define the <expression> value as well as the expression on the first line (<term> <mult_operator> <term>). This is what is meant by a rule. The (<expression>) expression may be replaced with the (<term> <mult_operator> <term>) expression. The entire sentence is considered a rule.

15.2 Literal Tokens

```
DIGIT          →   "0"-"9"
LETTER         →   "a"-"z" , "A"-"Z"
LETTER_V       →   "V | “v”
V_DIGIT        →   0 .. 255
V_DIGIT+1      →   1 .. 256
NUMBER         →   [ “-“ ]{DIGIT}  ["."] [ {DIGIT} ]
LPAREN         →   "("
RPAREN         →   ")"
DIV            →   "/"
```

15.3 RELATIONAL OPERATORS

```
GT             →   ">"
LT             →   "<"
GTEQ           →   ">="
LTEQ           →   "<="
EQ             →   "="
NE             →   "<>"
```

15.4 MATHEMATICAL OPERATORS

```
MUL            →   "*"
DIV            →   "/"
```

```
ADD             →   "+"
SUB             →   "-"
```

15.5 UNARY OPERATORS

```
NOT             →   "NOT"
MOD             →   "MOD"
ABS             →   "ABS"
SIN             →   "SIN"
COS             →   "COS"
TAN             →   "TAN"
EXP             →   "EXP"
DEG             →   "DEG"
RAD             →   "RAD"
ASIN            →   "ASIN"
ACOS            →   "ACOS"
ATAN            →   "ATAN"
RAN             →   "RAN"
LOG             →   "LOG"
SEC             →   "SEC"
ASEC            →   "ASEC"
```

15.6 TOKEN DELIMITERS

specifies end of one token and beginning of another

```
CRLF            →   %x0D %x0A ; actual ABNF code - ASCII
SPACE           →   %x20      ; assumes ASCII
HTAB            →   %x09      ; horizontal tab

DELIMITER       →   [ {SPACE } ]   CRLF   [ {SPACE} ]
                →   [ {SPACE } ]   HTAB   [ {SPACE } ]
                →   [ {SPACE } ]   LPAREN [ {SPACE } ]
                →   [ {SPACE } ]   RPAREN [ {SPACE } ]

Keywords =      {
          EXIT, RETURN, IF, THEN, ELSE, ENDIF, WHILE,
          ENDWHILE, LOOP, NEXT,
          SUBROUTINE, GOTO, INDEX, LINEAR, G0, G1,
          CW_CIRCLE, CCW_CIRCLE, G2, G3, G4, G40, G41, G42,
          G43, G44, G70, G71, G90, G91, G92,
          M47, G23, G24, SKEY, CYCLE, SCALE
          DWELL, M0, M1, VELOCITY, ROUNDING,
          PROGRAM, ENGLISH,
          METRIC, ABSOLUTE, INCREMENTAL, ON, OFF,
          ALL, CANCEL,
          INCREMENTAL, UNITS, SECOND, MINUTE, SOFTWARE,
          HOME,
          DISABLE, ENABLE, GET, SET, WAIT, SKEY, DISPLAY,
          SLEW, POSITION, MESSAGE, #FIRE,
```

```
#CENTERONFIDUCIAL, FAULTACKNOWLEDGE, $, SCF,
ACCELERATION, CYCLE, DWELL,
VELOCITY, DEFINE, UNDEFINE, PPI, CUTTER,
IMMEDIATE, ROTATION,SPLINE, OUTPUT, INPUT,
DAC, FL, FREERUN, GAIN, NOTCH, DEADBAND, CLAMP,
GEAR, HALT, LVDT, CW, CCW, MC, MSET,
PLANE, WAIT, START, HALT, ABORT, PLANE, QUEUE,
AGAIN, CANCEL, MAP, RAMP, TRAJECTORY, SEGMENT,
UMFO, ACCELERATION, AT, BOARD, BRAKE, FREERUN,
FL, GAIN, GEAR,
LVDT, MC, MSET, PLANE, PLC, QUEUE, RAMP, SEGMENT,
SLEW, ACCELERATION
TE, TD, TP, TRAJECTORY, UMFO, PSOP, PSOD, PSOF

#GRAPHICS, #DRAWING, #OVERLAY TYPE, #OVERLAY
COLOR, #ERODE, #DILATE, #SHARPEN, #BRIGHTEN,
#IMAGE SAVE, #IMAGE LOAD, #IMAGE REDRAW, #THRESH
AUTO, #THRESH BILEVEL, #FIND PATTERN, #FIND BLOB,
#FIND EDGE, #ROI TRAIN, #ROI HIDE, #ROI SHOW,
#ROI ADD,

}
```

15.7 Grammar Specification

In order to parse any language, the language must be described by a formal, context-free grammar. A formal grammar is a mathematical construct. The most common formal system for describing grammatical rules for humans to read is Backus-Naur Form or "BNF", which was developed in order to specify the language Algol 60. Any grammar expressed in BNF is a context-free grammar.

In the formal grammatical rules for a language, each kind of syntactic unit or group is named by a symbol. Those, which are built by grouping smaller constructs according to grammatical rules, are called non-terminal symbols; those, which can't be subdivided, are called terminal symbols or token types. We call a piece of input corresponding to a single terminal symbol a token, and a piece corresponding to a single non-terminal symbol a group. Rules are given for constructing the groups from their parts, as one or more syntactic groups are specified.

We can use the PPI language as an example of what symbols, terminal and non-terminal, mean. The terminal symbols, or tokens, of PPI are identifiers, constants (numeric), and the various keywords, arithmetic operators and punctuation marks. So the tokens of a grammar for PPI include `identifier', `number', plus one symbol for each keyword, operator or punctuation mark: `if', 'loop', 'while', 'endwhile', 'else', 'subroutine', `return', `open-bracket', `close-bracket', `comma' and many more. (These tokens can be subdivided into characters, but that is a matter of lexicography, not grammar.) By convention, tokens should be written in upper case to distinguish them from non-terminal symbols: for example, INTEGER, IDENTIFIER, IF, LOOP, WHILE, ELSE, SUBROUTINE, or RETURN. A terminal symbol that stands for a particular keyword in

the language should be named after that keyword converted to upper case. A terminal symbol can also be represented as a character literal, just like a C character constant. You should do this whenever a token is just a single character (parenthesis, plus-sign, etc.): use that same character in a literal as the terminal symbol for that token.

A non-terminal symbol in the formal grammar is represented as a group. A group is defined as a rule that is made up of other non-terminals and tokens. For example, in the PPI language, one kind of group is called an `expression'. One rule for making an expression might be, "An expression can be made of a “NOT” sign and another expression". Another would be, "An expression can be an integer". As can be seen, rules are often recursive; however there must be at least one rule which leads out of the recursion. By convention, all non-terminals are written in lower case, such as ‘expression’, ‘term’, or ’primary’.

The PPI context-free language is characterized as an LR(1). In brief, this means that it is possible to tell how to parse any portion of an input string with just a single token of look-ahead. The Translator was written however, to allow for additional look-ahead if the PPI grammar is changed so that it no longer meets the restrictions of a single token look-ahead grammar.

```
<file>          →     <program> {<subroutine>}

<subroutine>    →     <label> <statement_list> "RETURN"

<program>             →     "START" <statement_list> "EXIT"

<statement_list>      →     <statement> {<statement>}

<statement>           →     <labeled_statement>
                      →     <assignment_statement>
                      →     <selection_statement>
                      →     <iteration_statement>
                      →     <jump_statement>
                      →     <U500_statement>
                      →     <PPI_statement>
                      →     <comment_statement>
                      →     <blank_line_statement>
                      →     <null_statement>

<labeled_statement>   →     <label> [<statement>]

<label>                     →     “:” + Identifier

<assignment_statement>      →     Identifier  “=”  <expression>
                            →     Identifier = axis_position

<axis_positon>              →     "$" <axis> "FP"
                            →     "$" <axis> "RP"
                            →     "$" <axis> "AP"

<input_port>          →     "$" "IN"  V_DIGIT+1
                            →     "$" "INP" V_DIGIT+1

<axis>                →     "X" | "x"
                            →     "Y" | "y"
```

```
                              →     "Z" | "z"
                              →     "U" | "u"

<expression>          →       <term>  { <add_operator> <term> }

<add_operator>                →     "+"
                              →     "-"

<term>                →       <primary>  { <mul_operator> <primary> }

<mul_operator>                →     "*"
                              →     DIV

<unary>                       →     "NOT"<primary>
                              →     "ABS"<primary>
                              →     "SIN"<primary>
                              →     "COS"<primary>
                              →     "TAN"<primary>
                              →     "ASIN"<primary>
                              →     "ACOS"<primary>
                              →     "ATAN"<primary>
                              →     "EXP"<primary>
                              →     "LOG"<primary>
                              →     "RAN"<primary>
                              →     "SEC"<primary>
                              →     "ASEC"<primary>
                              →     "DEG"<primary>
                              →     "RAD"<primary>

<primary>                     →     Identifier
                              →     Number
                              →     <unary>
                              →     ( <expression> )
                              →     <input_port>
                              →     <axis_position>

<selection_statement>         →     "IF" <relational_expression> <label>
                              →     "IF" <input_expression> <label>
                              →     "IF"  <relational_expression>  "THEN"
                                          <statement_list>
                                    "ENDIF"
                              →     "IF"  <relational_expression>  "THEN"
                                          <statement_list>
                                    "ELSE"
                                          <statement_list>
                                    "ENDIF"

<relational_expression> → <expression> GT <expression>
                              →     <expression> LT <expression>
                              →     <expression> EQ <expression>
                              →     <expression> NE <expression>
                              →     <expression> LTEQ <expression>
                              →     <expression> GTEQ <expression>
                              →     <input_expression>
                              →     <expression>
                              →     (<expression> LT <expression>)
                              →     (<expression> EQ <expression>)
                              →     (<expression> NE <expression>)
                              →     (<expression> LTEQ <expression>)
                              →     (<expression> GTEQ <expression>)
                              →     (<input_expression>)
                              →     (<expression>)
```

```
<input_expression>  →     <input_port>  EQ  V_DIGIT+1

<iteration_statement>     →     "WHILE"   <relational_expression>
                                      <statement_list>
                                "ENDWHILE"
                          →     "LOOP" ( Integer )
                                      <statement_list>
                                "NEXT"
                          →     "LOOP" ( Identifier  )
                                      <statement_list>
                                "NEXT"

<jump_statement>          →    "GOTO" <label>
                          →    "SUBROUTINE" <label>

<comment_statement>       →     { <statement> } <comment>
<comment >                →     ";"  [ { LETTER + DIGIT } ]

<blank_statement>   →     DELIMITERS
```

16 Appendix C – PRG Programming Notes

1. EVERY SUBROUTINE MUST HAVE A STARTING LABEL AND A FINISHING RETURN STATEMENT - I.E. - EVERY SUBROUTINE MUST HAVE A SINGLE ENTRY POINT AND A SINGLE EXIT POINT

If there is no return statement for a subroutine or a command is used to exit the subroutine prematurely, the translator will assume that the next RETURN statement found is the RETURN statement for the previous subroutine. Things can get very confused that way. As MMI does not provide for multiple nested levels of subroutine calls, they do not need to keep a count of subroutine entry and exit points as the translator does. This was a judgement call by both Mike D. and myself due to the fact that everyone has always ended a subroutine with a RETURN statement.

2. EVERY PRG PROGRAM MUST USE AN EXIT STATEMENT (or m2) IN ORDER TO TELL THE TRANSLATOR THATANY STATEMENTS THAT FOLLOW ARE SUBROUTINES OR OTHER CODE THAT SHOULD NOT BE EXECUTED AS PART OF THE PROGRAM.

I initially also required a START command statement in order to allow subroutine definitions at the top of the file, however, since Mike and Greg's programs differed on their use of the START command to indictae PRG program beginning, I had to allow for both. Since the START command does not have to be used (it is specified in the Aerotech manual to be used) I can not allow the definition of subroutines or any other code before the actual PRG code to be executed. In order to provide for subroutines at all, they have to be somewhere in the file and there has to be a command statement that indicates where the PRG main code stops and the subroutines begin. This command is the EXIT command. It is required by all MMI and Translator programs for correct execution, although if the command is missing, both programs will continue execution beyond the expected end of the main PRG program and thus the execution will be unpredictable.

3. WHEN USING THE SUBROUTINE COMMAND, THE NAME OF THE SUBROUTINE TO JUMP TO MUST BE DEFINED. THE TRANSLATOR IS NOT CASE SENSITIVE IN THAT THE SUBROUTINE LABEL NO LONGER HAS TO BE THE EXACT ASCII CHARACTERS AS THE NAME OF THE SUBROUTINE THAT YOU WANT TO JUMP TO.

Correct:

```
SU:   MYSUB

:MYSUB
```

Now also correct

```
SU:   mysub

:MYSUB
```

This functionality was designed to be that way but can be changed if people have a problem with it. Of all the PRG programs that I have seen so far most people define their label and use the exact same characters to jump to that label. I don't know what MMI does here as no one has complained before.

4. ALWAYS USE THE G90 AND G91 COMMANDS ON THE SAME LINE AS THE LINEAR OR CCW COMMANDS. IF A G1 COMMAND DOES NOT ALSO HAVE THE G90 OR G91 SPECIFIED, IT IS SOMETIMES DIFFICULT TO KNOW WHAT STATE YOU ARE IN WHEN RETURNING FROM A SUBROUTINE
Most of our example programs do not yet follow this standard and we see problems because of it. As Potomac develops their Standards for PRG programming, they will be forwarded.

5. IF A SYNTAX ERROR OCCURS WHEN IT HAS BEEN DETERMINED THAT NO TRUE SYNTAX ERROR EXISTS THERE ARE SEVERAL ALTERNATIVES.
1) If the syntax error occurs on a line that has a control flow statement on it such as a LOOP, IF-THEN-ELSE, WHILE, etc., an alternate control statement should be tried in order to correct the error. If the incorrect syntax error is a bug in the PPI translator, it is probable that if the same control flow is coded in a different way (i.e. use a LOOP statement instead of a WHILE statement) the syntax error will not be recognized and you can continue with your PRG programming until Potomac can correct the bug.
2) If the syntax error occurs on a line that has a complex mathematical instruction (multiply nested expressions in parentheses) split the expression up into multiple lines. I.e. make the expression simpler.
3) If an error is occurring on every line even though it is known that no syntax error exists there is probably a comment that has begun with a colon instead of a semicolon and the translator is seeing it as a valid label statement. Also, check for line wrap. No lines can wrap - there must be a valid carriage return at the end of each line in order for the interpreter to tell that the line is a new line.

17 Index